Arquitetura e organização de computadores
Sistema de Computação

SIAC 202 - Arquitetura de Computadores

Prof.: Félix do Rêgo Barros

felixregobarros@gmail.com

Baseado em W. Stallings – Arquitetura e Organização de Computadores

MIPS (Microprocessor Without Interlocked Pipeline Stages)

(Microprocessador sem Estágios de Pipeline Integrados)

Exemplo didáticos em livros Starnford MIPS comercializado por MIPS Technologies (www.mips.com)
Ampla fatia de mercado de núcleos embarcados.

Aplicações em eletrônica de consumo, Impressoras.....
Típico de conjunto de instruções modernos.

Operações Aritméticas

Princípio de Projeto 1: simplicidade favorece a regularidade

- Regularidade torna implementação mais simples
- Simplicidade permite desempenho melhor a custo menor

Todas operações aritméticas tem a mesmas forma

- Três operandos: duas origens e um destino

Add a,b,c ➔ a = b + c

Registradores como operandos

Princípios de Projeto 2: menor é mais rápido

Instruções aritméticas usam registradores como operandos
MIPS tem um conjunto de 32 registradores de 32 bits
Numerados de 0 a 31

Dados de 32 bits são chamados: *word (PALAVRA)*

Registradores

$t0, $t1, ... $t9 ➔ valores temporários
$s0, $s1, ... $s7 ➔ variáveis a serem salvas

Registrador 0 ($zero) do MIPS é a constante 0
Não pode ser sobrescrita

Exemplo: mover conteúdo de registradores

Add $t2,$s1,$zero

Memória

Endereço por byte

Palavras (word) alinhadas na memória

Endereços precisam ser múltiplos de 4

É Big Endian: byte mais significativo no endereço menos significativo da palavra

Operandos em memória

Código em C:

g = h +[A];

g em $s1, h em $s2, endereço base de A em $s3

Código compilado MIPS:

Index 8 requer offcet de 32 (4 bytes por palavra)

```
Lw st0,32($s3)  # load word
Add $s1,$s2,$t0
```

Operandos imediatos

Princípios de Projeto 3: Faça o caso comum rápido

Constantes pequenas são comuns

Operandos imediatos evitam instrução de load

Constante especificada na própria instrução

Add $s3,$s3,4

Não existe instrução de subtração com imediato

Use constante negativa

Add $s2,$s1, -1

Operandos imediatos

Princípios de Projeto 3: Faça o caso comum rápido

Constantes pequenas são comuns

Operandos imediatos evitam instrução de load

Constante especificada na própria instrução

Add $s3,$s3,4

Não existe instrução de subtração com imediato

Use constante negativa

Add $s2,$s1, -1

Instrução MIPS

Considere a seguinte instrução em linguagem C:

a = b + c;

Essa instrução, para ser executada no processador, deve ser convertida para MIPS e depois para 0 e 1.

Para as conversões usamos os registradores de uso geral do MIPS. Normalmente, nas nossas conversões, as variáveis que armazenam valores usam os registradores de nome **t** e as variáveis que contêm os operandos usam os registradores de nome **s**.

A Tabela apresenta os 32 registradores do MIPS 32 bits.

Nome do Registrador	Número	Binário	Uso
$zero	0	000 000	constante zero
$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

Linguagem de Montagem

De acordo com a tabela, usaremos então os registradores que vão de $t0 à $t7 e de $s0 à $s7. A instrução a = b + c ficará da seguinte forma:

ADD $t0, $s0, $s1

ADD é o mnemônico para ADIÇÃO,
$s0 é o valor que está armazenado em **b**,
$s1 o valor da variável **c** e
$t0 é a variável **a.**

Essa é a instrução Assembly MIPS correspondente à instrução em linguagem C, também chamada de **Linguagem de Montagem.** Mas isso ainda não é suficiente para que a instrução seja executada no microprocessador. Os nomes dos registradores usados aqui foram escolhidos arbitrariamente, mas durante a execução real são usados os registradores que estiverem livres naquele instante.

OBS.: Mnemônico (memória) fácil de ser memorizado.

Linguagem de Máquina

O próximo agora é converter a instrução de Linguagem de Montagem para **Linguagem de Máquina**. Cada registrador tem um número, conforme apresenta a Tabela. A Linguagem de Máquina corresponde a trocar o NOME DO REGISTRADOR pelo seu NÚMERO. A instrução ficará da seguinte forma:

ADD $t0, $s0, $s1

ADD $8, $16, $17

Nome do Registrador	Número	Binário	Uso
$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

o próximo passo é fazer a REPRESENTAÇÃO da instrução. No MIPS existem três FORMATOS de instruções, veremos um agora, que é o **formato do tipo R** (registrador).

Formato de Instrução do Tipo R

op	rs	rt	Rd	Shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
opcode ou código da operação	registrador do primeiro operando fonte	registrador do segundo operando fonte	registrador do operando destino	Deslocamento	função de operação ou código de função

O primeiro campo, **OP**, é destinado para identificação do código de operação que será realizada. O último campo seleciona uma operação secundária, por exemplo, a operação principal é a aritmética, mas a secundária é uma soma. Nem todas as operações têm dois códigos (op-funct) e nesse caso o campo **FUNCT** é setado com o valor zero.

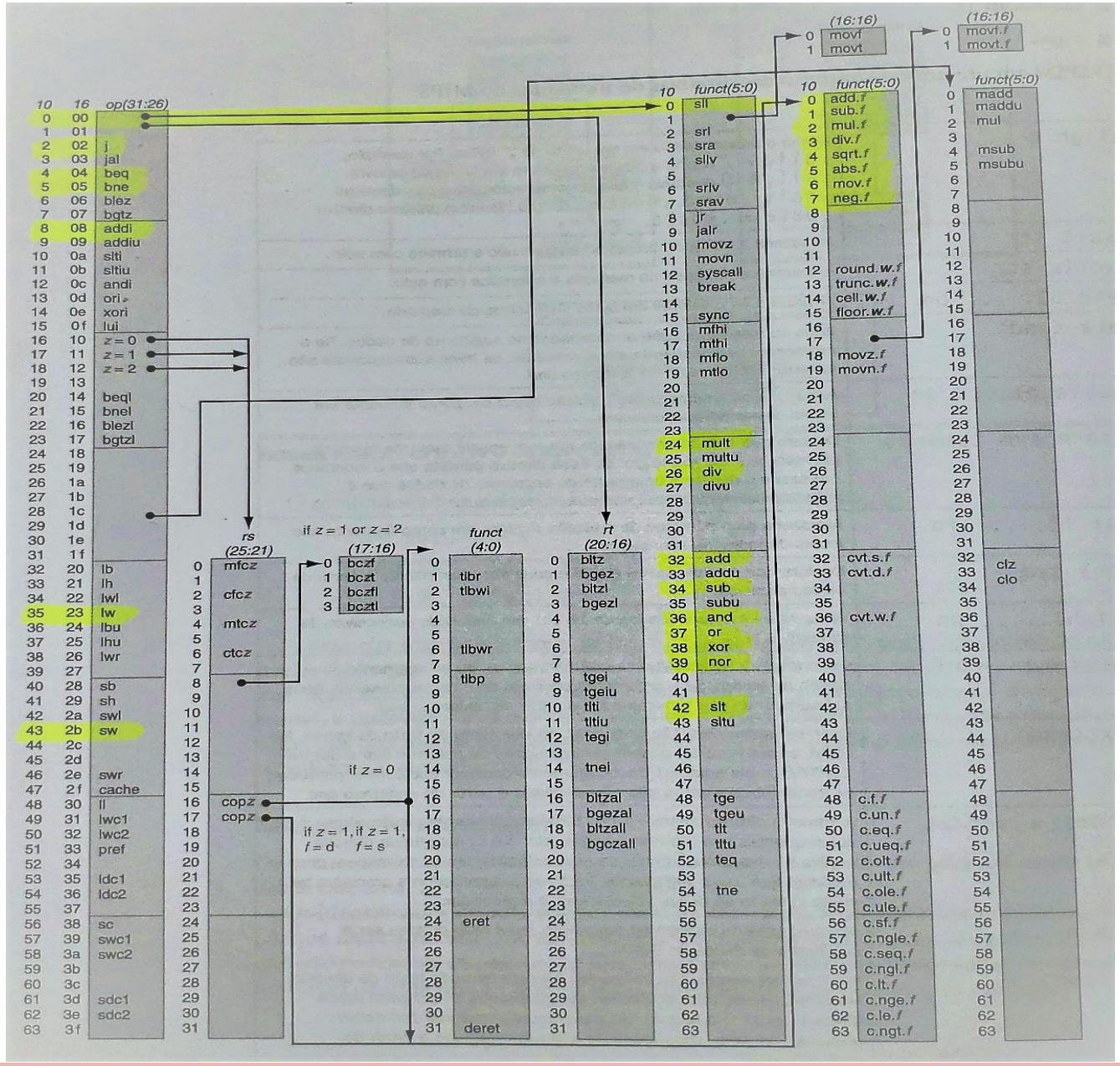
Os campos **RS** e **RT** são destinados aos operandos fontes, em ordem, da esquerda para a direita, conforme aparecem na operação. Já o campo **RD** é destinado para o armazenamento do resultado da operação, portanto, operando destino. **Shamt** é um campo usado para setar uma quantidade de deslocamento mas, por hora, não o usaremos, por isso será setado com o valor zero.

Precisamos também conhecer as operações e seus valores decimais correspondentes.
A Tabela 3 apresenta os valores binários e decimais para cada instrução que utilizaremos.

Tabela 3: Opcodes

OPCODE	DECIMAL	BINÁRIO
DIV	26	011 010
ADD	32	100 000
SUB	34	100 010
AND	36	100 100
OR	37	100 101
SLT	42	101 010
BNE	4	000 100
BEQ	5	000 101
J	2	000 010
JR	8	001 000
LW	35	100 011
SW	43	101 011

Tabela 3: Opcodes



Consultando a Tabela 3, a instrução ficará da seguinte forma:

op	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. A instrução está representada no seu formato específico, agora fica bem mais fácil pra chegar no **código 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.

passo a passo para realizar a conversão:

1. Converter a instrução de alto nível para Linguagem de Montagem;
2. Converter a instrução na Linguagem de Montagem para Linguagem de Máquina;
3. Fazer a representação correspondente da Linguagem de Máquina;
4. Converter a representação da Linguagem de Máquina para Código de Máquina.

Conversão de uma instrução com parênteses

Considere a seguinte instrução em linguagem C:

a = b - (c - d) + e

Como resolveríamos isso na mão, com uma folha de almaço? Sem saber os valores reais, e considerando a ordem de prioridade de execução das operações, a resolução dessa instrução ficaria assim:

a = b - ***(c - d)*** + e
a = ***b - T1*** + e
a = ***T3 + e***
a = **T4**

Onde **T1, T2, T3 e T4** são os valores resultantes e o negrito/itálico representa a parte da expressão matemática que está sendo resolvida naquele instante. Se você substituir b, c, d, e e com valores de números inteiros ou reais, chegará em um resultado final.

Linguagem de Montagem

Vamos fazer a conversão, começamos pela Linguagem de Montagem, lembrando sempre de consultar as tabelas da arquitetura do conjunto de instruções MIPS 32 bits. Como é uma instrução com parênteses, devemos primeiro resolver os parênteses, assim como em qualquer outra expressão matemática.

Depois dos parênteses, devemos voltar fazendo a leitura da expressão da ESQUERDA para a DIREITA, sem alterar a ordem em que os operandos aparecem na expressão. Então, primeiro o parênteses, depois **b - (c - d)** e somente depois somamos com **e**.
Usaremos os seguintes registradores: **a = $s0, b = $s1, c = $s2, d = $s3** e **e = $s4**.
Também precisaremos usar registradores temporários **$t** para poder armazenar os valores de cada parte da expressão.

Código MIPS correspondente à expressão a = b - (c - d) + e

```
1 SUB $t1, $s2, $s3     # $t1 = ( $s2 - $s3 ) é o mesmo que $t1 = ( c - d )
2 SUB $t2, $s1, $t1    # $t2 = $s1 - $t1 é o mesmo que $t2 = b - ( c - d )
3 ADD $s0, $t2, $s4     # $s0 = $t2 + $s4 é o mesmo que $s0 = b - ( c - d ) + e
```

Linguagem de Máquina

```
1 SUB $t1, $s2, $s3     # $t1 = ( $s2 - $s3 ) é o mesmo que $t1 = ( c - d )
2 SUB $t2, $s1, $t1    # $t2 = $s1 - $t1 é o mesmo que $t2 = b - ( c - d )
3 ADD $s0, $t2, $s4     # $s0 = $t2 + $s4 é o mesmo que $s0 = b - ( c - d ) + e
```

Ao invés de usarmos dois registradores diferentes, podemos usar apenas um, somente **$t1** e descartamos **$t2.** "Economizar" registradores temporários

```
1 SUB $t1, $s2, $s3     # $t1 = ( $s2 - $s3 ) é o mesmo que $t1 = ( c - d )
2 SUB $t1, $s1, $t1    # $t1 = $s1 - $t1 é o mesmo que $t1 = b - ( c - d )
3 ADD $s0, $t1, $s4     # $s0 = $t1 + $s4 é o mesmo que $s0 = b - ( c - d ) + e
```

A linguagem de máquina corresponde a trocar os nomes dos registradores pelo seu número, então nosso código ficará assim:

```
1 SUB $9, $18, $19     # $t1 = ( $s2 - $s3 ) é o mesmo que $t1 = ( c - d )
2 SUB $9, $17, $9     # $t1 = $s1 - $t1 é o mesmo que $t1 = b - ( c - d )
3 ADD $16, $9, $20     # $s0 = $t1 + $s4 é o mesmo que $s0 = b - ( c - d ) + e
```

Representação

A representação corresponde a pegar as nossas instruções MIPS e colocá-las no formato correspondente, conforme mostra a Tabela 1:

Tabela 1: Representação das instruções

op	rs	rt	rd	shamt	funct
0	18	19	9	0	34
0	17	9	9	0	34
0	9	20	16	0	32

Código de Máquina

Tabela 2: Código binário para cada campo do formato de instrução

op	rs	rt	rd	shamt	funct
000000	10010	10011	01001	00000	100010
000000	10001	01001	01001	00000	100010
000000	01001	10100	10000	00000	100000

Assim, o código de máquina final é o seguinte:

00000010010100110100100000100010
00000010001010010100100000100010
00000001001101001000000000100000

Exercícios

1. *Converta as instruções em C abaixo para instruções MIPS. Considere os seguintes registradores para cada variável: a = $s0, b = $s1, c = $s2, d = $s3, e = $s4, f = $s5.*

1. a = b - c
2. b = a + c
3. d = (a + b - c)
4. f = (a + b) - d
5. c = a - (b + d)
6. e = (a - (b - c)
7. e = (a - (b - c) + f)
8. f = e - (a - b) + (b - c

converter uma instrução com Array no MIPS.

Como exemplo, suponha a seguinte instrução em linguagem C:

a = b + c[10];

Vamos usar o registrador **$s0** para a variável **a**, **$s1** para variável **b** e **$s2** para a variável **c**. **c** é um vetor e não sabemos quantas posições ele tem, só sabemos que iremos somar o valor da variável **b** com o valor que está armazenado na posição 10 do vetor **c**.

pra podermos fazer a conversão dessa instrução introduzirei uma nova instrução que também é de um tipo diferente do que já vimos até aqui.

converter uma instrução com Array no MIPS.

Instruções de Formato Tipo I

As instruções de formato Tipo I têm menos campos que as instruções do formato tipo R (aritméticas), mas ainda tem 32 bits, não vamos nos esquecer de que no MIPS todas as instruções tem exatamente o mesmo tamanho em bits, ok?! A instrução de formato Tipo I é representada na Tabela e normalmente são classificadas como instruções de transferência de dados.

Tabela: Representação de Instruções do Formato Tipo I

opcode	rs	rt	endereço
6 bits	5 bits	5 bits	16 bits
código de operação	registrador destino	registrador fonte	endereço de memória

converter uma instrução com Array no MIPS.

Instruções de Formato Tipo I

LOAD WORD

Essa instrução transfere dados da memória para os registradores e, sempre que tivermos um Array, deveremos utilizá-la pois, antes de manipularmos o valor de uma determinada posição do Array, devemos tê-lo disponível para isso. Sua sintaxe é a seguinte:

LW registrador_destino, valor (registrador_fonte)

Exemplo:

LW $t0, 30 ($s0) # $t0 = memória [$s0 + 30]

converter uma instrução com Array no MIPS.

Instruções de Formato Tipo I

LOAD WORD

```
LW $t0, 30 ( $s0 )     # $t0 = memória [ $s0 + 30 ]
```

O registrador **$t0** receberá o valor que está no endereço de memória que é calculado pela própria instrução: **$s0 + 30**.

Então, toda vez que você usar a instrução LW, você está transferindo para um registrador, um valor que está no endereço de memória calculado pela soma do registrador fonte com um valor. Neste exemplo é um valor dado (30), ou seja, é a posição 30 do Vetor que aqui é representado por $s0.

converter uma instrução com Array no MIPS.

LOAD WORD

Compilação de uma atribuição com um operando na memória

Agora que já fomos apresentados às instruções de formato TIPO I e também à instrução LW, vamos ver como fica a conversão da nossa instrução. O primeiro passo é converter c[10] que ficará assim:

```
LW $t0, 10 ($s2)    # $t0 = memória [ $s2 + 10 ]
```

Observe que **$s2** é o vetor **c**, 10 é a posição do Vetor e **$t0** é um registrador temporário que armazenará o valor que está em **c[10]**. O segundo passo é fazer b + c[10]:

```
ADD $s0, $s1, $t0     # $s0 = $s1 + $t0
```

Em que **$t0** é o valor de **c[10]**, **$s0** é a variável **a** e **$s1** é a variável **b**. Assim, o código final para a = b + c [10] fica da seguinte forma:

```
LW $t0, 10 ( $s2 )
ADD $s0, $s1, $t0
```

Observe também que uma instrução da linguagem C foi convertida em duas instruções para linguagem MIPS.

converter uma instrução com Array no MIPS.

LOAD WORD

Linguagem de Máquina

A Linguagem de Máquina para a = b + c [10] ficará da seguinte forma:

LW $8, 10 ($18)

ADD $16, $17, $8

Representação da Linguagem de Máquina

op	rs	rt	rd	shamt	funct
35	8	18	10		
0	17	8	16	0	32

Código de Máquina

O código de máquina para a = b + c [10] ficará da seguinte forma:

opcode	rs	rt	rd	shamt	funct
100 011	01000	10010	0000 0000 0000 1010		
000 000	10001	01000	10000	00000	100 000

10001101000100100000000000001010
00000010001010001000000000100000

converter uma instrução com Array no MIPS.

LOAD WORD

Exercícios

Converta as instruções abaixo:

1.a = b[15] - c;
2.b = a[5] + c[3];
3.c = b - a[21];

Use $s0 para a, $s1 para b e $s2 para c.

converter uma instrução com Array no MIPS.

STORE WORD - SW

A instrução **Store Word** tem como objetivo armazenar um valor, que está em um registrador, na memória. Store Word significa **Armazenar Palavra**, ao pé da letra, assim como Load Word significa **Carregar Palavra.**

Essas duas instruções, LW e SW, são idênticas em seu formato, o que muda de fato é o objetivo de cada uma, de forma que devemos tomar cuidado ao passá-las para a Representação, pois os campos têm de receber os valores corretos, cuidado para não se confundir. Veja a Tabela

Tabela 1: Representação de Instruções do Formato Tipo I

opcode	rs	rt	endereço
6 bits	5 bits	5 bits	16 bits
código de operação	registrador destino	registrador fonte	endereço de memória

converter uma instrução com Array no MIPS.

Tabela: Representação de Instruções do Formato Tipo I

opcode	rs	rt	endereço
6 bits	5 bits	5 bits	16 bits
código de operação	registrador destino	registrador fonte	endereço de memória

É exatamente igual à instrução LW, mas a SW inverte RS (fonte) e RT (destino)! Agora vamos ver a sintaxe da instrução:

SW registrador_fonte, valor (registrador_destino)

Exemplo:

SW $t0, 30 ($s0) # memória [$s0 + 30] = $t0

Exatamente a mesma forma, MAS, observe que agora a memória é quem está recebendo o valor do registrador; o registrador destino está depois e o registrador fonte está antes, isto é, estão invertidos em relação à instrução LW. Portanto, **$t0** é o valor que deve ser armazenado no endereço de memória determinado por **($s0 + 30)**.

Converter uma instrução com Array no MIPS.

Compilação de instruções SW

Agora que já fomos apresentados à instrução SW, vamos ver como fica a conversão da seguinte instrução:

a[15] = b + c

Vamos usar **$s0** para **a**, **$s1** para **b** e **$s2** para **c**. O primeiro passo é converter b + c, que ficará assim:

```
ADD $t0, $s1, $s2        # $t0 = $s1 + $s2
```

A instrução acima realiza a soma de b com c, armazenando o resultado em $t0. Agora, vamos armazenar $t0 no endereço de memória, que é o nosso array a[15]:

```
SW $t0, 15 ( $s0)      # memória [ 15 + $s0 ] =  $t0
```

a[15] é correspondente ao código 15 ($s0) e $t0 é o valor da soma. Portanto, o código final é:

```
ADD $t0, $s1, $2
SW $t0, 15 ( $s0 )
```

Converter uma instrução com Array no MIPS.

Linguagem de Máquina

A Linguagem de Máquina para **a[15] = b + c** ficará da seguinte forma: (lembre-se de consultar as tabelas que foram apresentadas no artigo PRIMEIRA INSTRUÇÃO MIPS)

ADD $8, $17, $18
SW $8, 15 ($16)

Representação da Linguagem de Máquina

A representação da linguagem de máquina para **a[15] = b + c** ficará da seguinte forma. RS e RT são os registradores fonte e RD o registrador destino, para as instruções do tipo R. **Para as instruções STORE WORD, RS é o registrador fonte e RT é o registrador destino.** Finalizando, para as instruções LOAD WORD, RS é o registrador destino e RT é o registrador fonte. Não se esqueça de consultar as tabelas para fazer a Representação.

opcode	rs	rt	rd	shamt	funct
0	17	18	8	0	32
43	8	16	15		

Código de Máquina

O código de máquina para **a[15] = b + c** ficará da seguinte forma:

opcode	rs	rt	rd	shamt	funct
000 000	10001	10010	01000	00000	100 000
101 011	01000	10000	0000 0000 0000 1111		

00000010001100100100000000100000
10101101000100000000000000001111

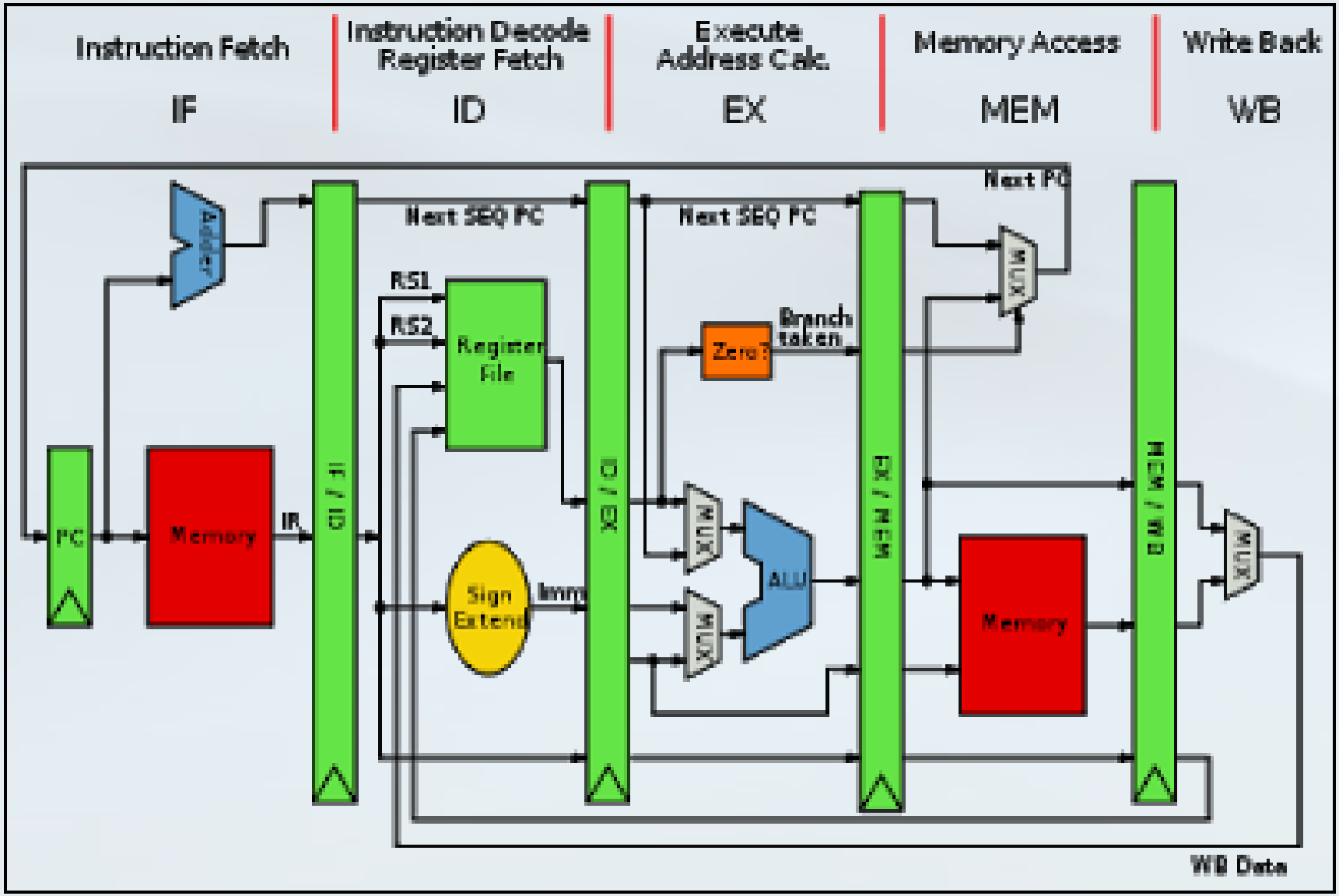
Converter uma instrução com Array no MIPS.

Exercícios

Converta as instruções abaixo:

1.a[10] = b - c;
2.b[245] = a + c;
3.c[0] = b - a;

Use $s0 para a, $s1 para b e $s2 para c.


Instruction Fetch
IF
Instruction Decode
Register Fetch
ID
Execute
Address Calc.
EX
Memory Access
MEM
Write Back
WB
Adder
Next PC
Next SEQ PC
Next SEQ PC
RS1
RS2
Register
File
Zero?
Branch
taken
MUX
PC
Memory
IR
IF / ID
ID / EX
EX / MEM
MEM / WB
Sign
Extend
Imm
ALU
Memory
WB Data