elainececiliagatto@gmail.com

MIPS 32 BITS CHEAT SHEET
Tipos de Instruções

Instruções Tipo R

opcode	rs	rt	rd	Shamt	Funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Código da operação	Registrador Fonte	Registrador Fonte	Registrador Destino	Deslocamento	Código de operação secundário

Instrução C	a = b + c;
Linguagem de Montagem	ADD $t0, $s0, $s1
Linguagem de máquina	ADD $8, $16, $17
Representação	0 16 17 8 0 32
Código de Máquina	000000 10000 10001 01000 00000 100000

Instruções Tipo I

opcode	rs	rt	endereço
6 bits	5 bits	5 bits	16 bits
Código da Operação	Registrador	Registrador	Endereço de memória

Instrução C	g = h + a[8];
Linguagem de Montagem	LW $t0, 8 ($s3) ADD $s1, $s2, $t0
Linguagem de Máquina	LW $8, 8 ($19) ADD $16, $17, $8
Representação	35 8 19 8 0 17 8 16 0 32
Código de Máquina	100011 010000 10011 0000 0000 0000 1000 000000 10001 01000 10000 00000 100000

Instruções Tipo J

opcode	rs	endereço
6 bits	5 bits	21 bits
Código da operação	Registrador	Endereço de memória

Instrução C	EXIT, ELSE, ou algum outro label, ou retorno de função
Linguagem de Montagem	JR $t0
Linguagem de máquina	JR $8
Representação	0 8 8
Código de Máquina	000000 01001 00000 00000 00000 001000

elainececiliagatto@gmail.com

MIPS 32 BITS CHEAT SHEET
Registradores e Memória

opcode	Operação básica da instrução
rs	Registrador do primeiro operando fonte
rt	Registrador do segundo operando fonte
rd	Registrador do operando destino
shamt	Shift amount, em português, quantidade de deslocamento. Utilizado em algumas instruções lógicas
funct	Função ou código de função. É um código variante do opcode. Utilizado em algumas instruções aritméticas.

MEMÓRIA

FRAME DE PILHA

Registrador	Decimal	Binário	Uso
$t0	8	001 000	Registradores temporários. Não preservados pela chamada.
$t1	9	001 001	
$t2	10	001 010	
$t3	11	001 011	
$t4	12	001 100	
$t5	13	001 101	
$t6	14	001 110	
$t7	15	001 111	
$s0	16	010 000	Registradores temporários salvos. Preservados pela camada.
$s1	17	010 001	
$s2	18	010 010	
$s3	19	010 011	
$s4	20	010 100	
$s5	21	010 101	
$s6	22	010 110	
$s7	23	010 111	

- 32 registradores de propósito geral;

- HI e LO: registradores de propósito especial para guardar resultados de divisão e multiplicação de inteiros e outras operações acumuladoras;

- PC: counter program, ou contador de programa, registrador de propósito especial;

Registrador	Decimal	Binário	Uso
$zero	0	000 000	Constante zero
$at	1	000 001	Montador
$v0	2	000 010	Avaliação de expressão e resultados de função
$v1	3	000 011	
$a0	4	000 100	Argumentos de Função
$a1	5	000 101	
$a2	6	001 110	
$a3	7	000 111	
$k0	26	011 010	Reservado para o Kernel do S.O.
$k1	27	011 011	
$gp	28	011 100	Global pointer
$sp	29	011 101	Stack pointer
$fp	30	011 110	Frame pointer
$ra	31	011 111	Endereço de retorno da função

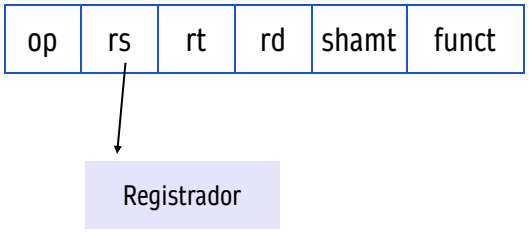
elainececiliagatto@gmail.com

MIPS 32 BITS CHEAT SHEET
Modos de Endereçamento e Instruções

Imediato

O operando é uma constante dentro da própria instrução.

Registrador

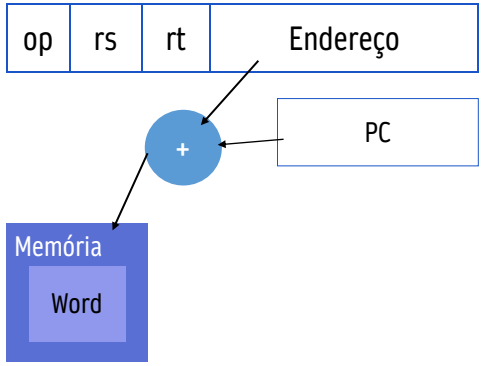


O operando é um registrador

- Instruções MIPS possuem endereços em BYTES
- Os endereços das palavras sequenciais diferem em 4
- MIPS é Big Endian

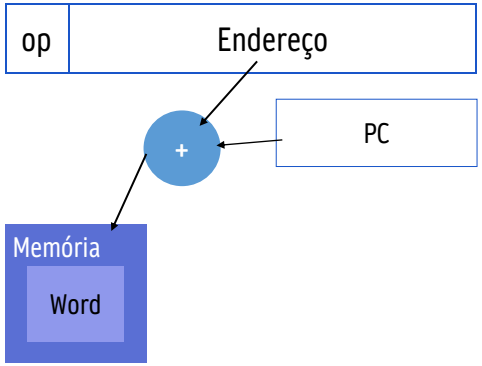
20 000
20 004
20 008
20 012
20 016

Relativo ao PC



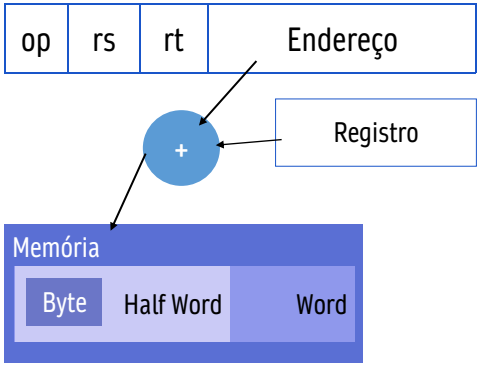
O endereço de desvio é a soma do PC e uma constante na instrução

Pseudodireto



O endereço de jump são os 26 bits da instrução concatenados com os bits mais altos do PC

Base



O operando está no local de memória cujo endereço é a soma de um registrador e uma constante na instrução

Instruções Implementadas

- Aritméticas, Lógicas e Relacionais;
- Desvios condicionais e incondicionais;
- Manipulação de operandos constantes e imediatos;
- Tratamento de Exceções e Interrupções;
- Carga e Armazenamento;
- Transferência de dados;
- Ponto Flutuante.

Principais Instruções

Instrução	Código
Add	$(32)_{10}$ $(20)_{16}$
addi	$(08)_{10}$ $(08)_{16}$
addiu	$(09)_{10}$ $(09)_{16}$
And	$(36)_{10}$ $(24)_{16}$
beq	$(04)_{10}$ $(04)_{16}$
bne	$(05)_{10}$ $(05)_{16}$
div	$(26)_{10}$ $(1A)_{16}$
divu	$(27)_{10}$ $(1B)_{16}$
j	$(02)_{10}$ $(02)_{16}$
jr	$(8)_{10}$ $(8)_{16}$
lw	$(35)_{10}$ $(23)_{16}$
mult	$(24)_{10}$ $(18)_{16}$
Nor	$(39)_{10}$ $(27)_{16}$
Or	$(37)_{10}$ $(25)_{16}$
sll	$(0)_{10}$ $(0)_{16}$
Slt	$(42)_{10}$ $(2A)_{16}$
sra	$(2)_{10}$ $(2)_{16}$
Sub	$(34)_{10}$ $(22)_{16}$
subu	$(35)_{10}$ $(23)_{16}$
sw	$(43)_{10}$ $(2B)_{16}$
Xor	$(38)_{10}$ $(26)_{16}$

elainececiliagatto@gmail.com

MIPS 32 BITS CHEAT SHEET
BNE, BEQ, SLT e SLTI

Passo a passo: conversão ALTO NÍVEL para ASSEMBLY

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.

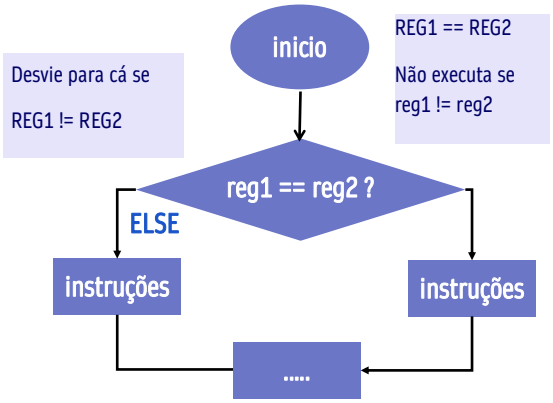
Passo a passo: conversão ASSEMBLY para ALTO NÍVEL

1. Converter o Código de Máquina para a Representação;
2. Converter a Representação em Linguagem de Máquina;
3. Converter a Linguagem de Máquina para Linguagem de Montagem;
4. Converter Linguagem de Montagem para instrução de alto nível;

BNE: BRANCH IF NOT EQUAL

Desvie se não for igual. Testa uma desigualdade. A próxima instrução a ser executada é aquela que estiver armazenada no endereço do LABEL se o valor no registrador 1 não for igual ao valor no registrador 2.

BNE REG1, REG2, ELSE # vá para ELSE se REG1 != REG2



Cálculo do endereço relativo da memória

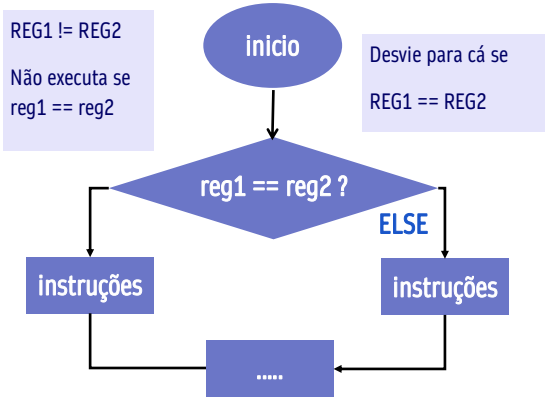
- Instruções de desvio acrescentam X palavras, ou X bytes, ao endereço da instrução seguinte à si mesma;

- Especificam o destino do desvio em relação à instrução seguinte e não em relação à instrução de desvio ou ao uso do endereço de destino completo

BEQ: BRANCH IF EQUAL

Desvie se for igual. Testa uma igualdade. A próxima instrução a ser executada é aquela que estiver armazenada no endereço do LABEL se o valor no registrador 1 for igual ao valor no registrador 2.

BEQ REG1, REG2, ELSE # vá para ELSE se REG1 == REG2



Exemplo:

```
80000 bne $s3, $s4, Else
80004 add $s0, $s1, $s2      # instrução seguinte à bne
80008 j Exit                 # + 2 palavras ou 8 bytes
                             # 80008 + 4 = 80012
80012 ELSE: sub $s0, $s1, $s2
80016 Exit:                  # endereço completo 20004 * 4
```

SLT: SET ON LESS THAN

Compara dois valores de dois registradores diferentes. Atribui o valor 1 a um terceiro registrador se o valor do primeiro registrador for menor que o valor do segundo registrador. Caso contrário, atribui zero.

SLT REG3, REG1, REG2

REG3 = 1 se REG1 < REG2

REG3 = 0 se REG1 > REG2

SLTI: SET ON LESS THAN IMMEDIATE

Compara o valor de um registrador com um valor constante ou imediato. Atribui o valor 1 a um terceiro registrador se o valor do primeiro registrador for menor que o valor do segundo registrador. Caso contrário, atribui zero.

SLTI REG3, REG1, VALOR

REG3 = 1 se REG1 < VALOR

REG3 = 0 se REG1 > VALOR

elainececiliagatto@gmail.com

MIPS 32 BITS CHEAT SHEET
LOAD WORD e STORE WORD

LOAD WORD

Transfere dados da memória para o registrador

LW REG1, valor (REG2) # REG1 = memória [REG2 + valor]

EXEMPLO 1: LOAD

Instrução:

g = h + a[8];

* multiplicar 8 por 4 devido à restrição de alinhamento

Linguagem de montagem:

```
LW $t0, 32($s3)
ADD $s0, $s1, $t0
```

Linguagem de máquina:

```
LW $8, 32 ($19)
ADD $16, $17, $8
```

Representação:

```
35 8 19 32
0 17 8 16 0 32
```

Código de Máquina:

```
100011 01000 10011 00000 00000 100000
000000 10001 01000 10000 00000 100000
```

STORE WORD

Transfere dados do registrador para a memória

SW REG1, valor (REG2) # memória [REG2 + valor] = REG1

EXEMPLO 2: STORE

Instrução:

g[8] = h + a;

* multiplicar 8 por 4 devido à restrição de alinhamento

Linguagem de montagem:

```
ADD $t0, $s1, $t0
SW $t0, 32($s0)
```

Linguagem de máquina:

```
ADD $8, $17, $8
SW $8, 32 ($16)
```

Representação:

```
0 8 17 8 0 32
43 8 16 32
```

Código de Máquina:

```
000000 01000 10001 01000 00000 100000
101011 01000 01000 00000 00000 100000
```

EXEMPLO 3: LOAD E STORE

Instrução:

g[2] = h + a[4];

Linguagem de montagem:

```
LW $t0, 16 ($s0)
ADD $s2, $s1, $t0
SW $t1, 4 ($s2)
```

Linguagem de máquina:

```
LW $8, 16 ($16)
ADD $18, $17, $8
SW $9, 4 ($18)
```

Representação:

```
35 8 16 16
0 8 18 17 0 32
43 9 18 4
```

Código de Máquina:

```
100011 01000 10000 00000 00000 10000
000000 01000 10010 10001 00000 100000
101011 01001 10010 00000 00000 000100
```

EXEMPLO 4: ÍNDICE VARIÁVEL

Instrução:

g = h + a[i];

Linguagem de montagem:

```
ADD $t0, $s3, $s3   # 2*i
ADD $t0, $t0, $t0   # 4*i
ADD $t0, $t0, $s2   # a[i]=(4*i+$s2)
LW $t1,0($t0)       # $t1=a[i]
ADD $s0, $s1, $t1   # g=h+a[i]
```

Linguagem de máquina:

```
ADD $8, $19, $19    # 2*i
ADD $8, $8, $8      # 4*i
ADD $8, $8, $18     # a[i]=(4*i+$s2)
LW $9,0($8)         # $t1=a[i]
ADD $16, $17, $9    # g=h+a[i]
```

Representação:

```
0 19 19 8 0 32
0 8 8 8 0 32
0 8 18 8 0 32
35 9 8 0
```

elainececiliagatto@gmail.com

MIPS 32 BITS CHEAT SHEET
Controle de Programa e Array

IF SIMPLES

Instrução: if (a==b){ c = a + b }

Resolução:

Se a ==b então execute a instrução c = a + b, caso contrário, sai do if. O desvio só vai acontecer se a ! = b, caso contrário não tem desvio!!! Portanto, Se (a==b) entra no if e se (a!=b) vai para EXIT (desvia).

Linguagem de Montagem:

```
BNE $s0, $s1, EXIT
ADD $t0, $s0, $s1
J EXIT
```

Linguagem de Máquina:

```
BNE $16, $17, EXIT
ADD $8, $16, $17
J EXIT
```

Representação:

```
5 16 17 EXIT
0 16 17 8 0 32
2 [endereço]
```

Código:

```
000101 10000 10001 [endereço]
000000 10000 10001 01000 00000 100000
000010 [endereço]
```

IF COMPOSTO

Instrução: if (a==b) { c = a + b } else { c = a — b}

Resolução:

Se a ==b então execute a instrução c = a + b, caso contrário, execute a instrução c = a—b.

Linguagem de Montagem:

```
BNE $s0, $s1, ELSE
ADD $t0, $s0, $s1
J EXIT
ELSE SUB $t0, $s0, $s1
```

Linguagem de Máquina:

```
BNE $16, $17, EXIT
ADD $8, $16, $17
J EXIT
ELSE SUB $8, $16, $17
```

Representação:

```
5 16 17 EXIT
0 16 17 8 0 32
2 [endereço]
ELSE 0 16 17 8 0 34
```

Código:

```
000101 10000 10001 [endereço]
000000 10000 10001 01000 00000 100000
000010 [endereço]
ELSE 000000 10000 10001 01000 00000 100010
```

FOR

Instrução:

```
for(indice=0; indice<10; indice++) {
    soma = Vetor[indice] + soma; }
```

Resolução:

```
LOOP:
# t0 = 0 se $s0 >= $s3 ( i >= n), t0 = 1 caso contrário
SLT $t0, $s0, $s3
 # se $s0 >= $s3 ( i >= n) vá para EXIT
BEQ $t0, $zero, EXIT
# $t1 = 4 * i, ou 4 * $s0
SLL $t1, $s0, 2
# t2 = ( vetor + ( 4 * i) )
ADD $t2, $s4, $t1
# $t3 = vetor[i], carregando o elemento do índice i
LW $t3, 0($t2)
# somando os elementos (soma = soma + vetor[i]
ADD $s1, $s1, $t3
# $s0 = $s0 + 1 (ou i = i + 1) é o contador
ADDI $s0, $s0, 1
# volta para o LOOP  EXIT
J LOOP
```

WHILE

Instrução:

```
while(save[i] == k) {  i += 1; }
```

Resolução:

```
LOOP:
    SLL $t1, $s3, 2        # $t1 = 4 * i
    ADD $t1, $t1, $s6      # $t1 = (4i + $s6)
    LW $t0, 0 ($t1)        # $t0 = save[i]
    # vá para EXIT se save[i] diferente de k
    BNE $t0, $s5, EXIT
    ADDI $s3, $s3, 1  # $s3 = $s3 + 1 (ou i = i + 1)
    J LOOP             # volta para o LOOP
    EXIT:
```

ARRAY

Instrução

```
int c[15] = {3, 0, 1, 2, -6, -2, 4, 10, 3, 7, 8, -9, -15, -20, -87, 0};
int a = 0, b = 30;
a = b + c[10];
```

Resolução:

```
.data
c: .word 3, 0, 1, 2, -6, -2, 4, 10, 3, 7, 8, -9, -15, -20, -87, 0
.text
# determinando o valor para $s1
LI $s1, 30
# colocando o endereço do array em $s2
LA $s2, c
# colocando o índice do array em $t2
LI $t2, 10
ADD $t2, $t2, $t2      # “2i”
ADD $t2, $t2, $t2      # “4i”
# combinando os dois componentes do endereço
ADD $t1, $t2, $s2
# obtendo o valor da célula do array
LW $t0, 0 ( $t1 )
# executando a soma
ADD $s0, $s1, $t0
```

elainececiliagatto@gmail.com

MIPS 32 BITS CHEAT SHEET

Controle de Programa e Procedimentos

SWTICH/CASE

Instrução:

```
switch (k) {
    case 0:
        f = i + j; // k = 0     break;
    case 1:
        f = g + h; // k=1     break;
    case 2:
        f = g – h; // k = 2     break;
    case 3:
        f = i – j; // k = 3     break;
}
```

Resolução:

```
.data
    jTable: .word L0,L1,L2,L3
.text
    # Definindo as variáveis: carregando valores para testar o código!
    LI $s1, 15    # g = $s1 = 15
    LI $s2, 20    # h = $s2 = 20
    LI $s3, 10    # i = $s3 = 10
    LI $s4, 5     # j = $s4 = 5
    LI $s5, -1    # k = $s5 = 2
    # $t4 = base address of the jump table
    LA $t4, jTable
    # Verificando os limites dos casos
    SLT $t3, $s5, $zero
    BNE $t3, $zero, EXIT
    SLTI $t3, $s5, 4
    BEQ $t3, $zero, EXIT
    # Calculando o endereço correto do Label
    SLL $t1, $s5, 2
    ADD $t1, $t1, $t4
    LW $t0, 0($t1)
    JR $t0 # Seleção do Label
    # Casos
L0: ADD $s0, $s3, $s4
    J EXIT
L1: ADD $s0, $s1, $s2
    J EXIT
L2: SUB $s0, $s1, $s2
    J EXIT
L3: SUB $s0, $s3, $s4
EXIT:
```

PROCEDIMENTOS

Instrução:

```
int exemplo ( int g, int h, int i, int j) {
   int f;
   f = ( g + h ) - ( i + j );
   return f;
}
```

Resolução:

```
.text
    LI $a0, 15      # g = $a0 = 15
    LI $a1, 20      # h = $a1 = 20
    LI $a2, 10      # i = $a2 = 10
    LI $a3, 5       # j = $a3 = 5
EXEMPLO:
# salva os registradores temporários usados pelo corpo da função
    ADDI $sp, $sp, -12
    SW $t1, 8 ($sp)
    SW $t0, 4 ($sp)
    SW $s0, 0 ($sp)
# corpo da função
    ADD $t0, $a0, $a1     # (g + h) = 15 + 20 = 35
    ADD $t1, $a2, $a3     # (i + j) = 10 + 5 = 15
    SUB $s0, $t0, $t1     # (g + h ) - ( i + j) = 35 – 15 = 5
    ADD $v0, $s0, $zero   # retorno da função f ($v0 = $s0 + 0 )
    LW $s0, 0 ($sp)       # restaura o registrador para o caller
    LW $t0, 4 ($sp)       # restaura o registrador para o caller
    LW $t1, 8 ($sp)       # restaura o registrador para o caller
    ADDI $sp, $sp, 12     # ajusta a pilha para excluir os 3 itens
    JR $ra                # volta para o endereço de retorno
```

Registradores para manipular **procedimentos:**

- $a0 até $a3: são os registradores de argumentos utilizados para a passagem de parâmetros;

- $v0 e $v1: são os registradores de valor utilizados para o retorno do procedimento;

- $ra: é o registrador de endereço do retorno do procedimento, utilizado na volta ao ponto de origem da chamada do procedimento.

JAL: é uma instrução de salto (jump) utilizada unicamente para os procedimentos (jump and link). Essa instrução desvia para um endereço e, ao mesmo tempo, salva o endereço da instrução seguinte no registrador de endereço de retorno.

JR: uma instrução de desvio incondicional para o endereço especificado em um registrador; ela volta ao endereço de retorno correto que é armazenado em $ra.

CALLER: é o programa que chama o procedimento, fornecendo os valores dos parâmetros.

CALLEE: é um procedimento que executa uma série de instruções armazenadas com base nos parâmetros fornecidos pelo Caller e depois retorna o controle para o Caller novamente.

SPILLED REGISTERS: é o processo de colocar variáveis menos utilizadas na memória

PILHA: gerencia as chamadas e retornos de procedimentos

STACK POINTER: é um valor que indica o endereço alocado mais recentemente em uma pilha, mostrando onde devem ser localizados os valores antigos dos registradores e onde os Spilled Registers devem ser armazenados.

PUSH: coloca palavras para cada registrador salvo ou restaurado na pilha. Valores são colocados na pilha pela subtração do valor do Stack Pointer.

POP: remove palavras da pilha. Valores são retirados da pilha pela soma do valor do stack pointer.