Arquitetura MIPS

A arquitetura MIPS é baseada num conjunto de instruções de comprimento fixo, regularmente codificado e usa um modelo de dados de load/store, onde todas as operações são realizadas em registos do processador e memória principal é acedida apenas por instruções de leitura e escrita. O modelo load/store reduz o número de acessos à memória, Este abrandamento requisitos de largura de banda de memória, simplifica o conjunto de instruções, e torna mais fácil aos compiladores a otimização da alocação de registos.

Instruções MIPS

Descrição do conjunto de instruções MIPS, seus significados, sintaxe, semântica e codificação bit codificações. A sintaxe refere-se à sintaxe da linguagem assembly. Hífens na codificação indicam bits “dont't care”que não são considerados quando uma instrução está a ser decodificada.

Registos de uso geral (general purpose registers, GPRs) são indicados com um sinal de dólar ($). As palavras SWORD e UWORD referem-se a tipos de dados de 32 bits com sinal e sem sinal, respectivamente.

A forma como o processador executa uma instrução e avança o seu contador de programa (program counter, PC) é a seguinte:

1. fetch da instrução apontada por PC
2. PC += 4
3. executar a instrução

Nota importante

- **TODOS os valores aritméticos imediatos são com extensão de sinal**. São depois tratados como números 32 bits com ou sem sinal, dependendo da instrução (a única diferença entre as instruções com e sem sinal é que as instruções com sinal podem gerar uma exceção de overflow e as instruções sem sinal não).

ADD – Add (com overflow)

Descrição	Adiciona dois registos e armazena o resultado num registo
Operação	`$d = $s + $t;`
Sintaxe	`add $d, $s, $t`
Codificação	`0000 00ss ssst tttt dddd d000 0010 0000`

ADDI – Add immediate (com overflow)

Descrição	Adiciona um registo e um valor imediato com extensão de sinal e armazena o resultado num registo
Operação	`$t = $s + imm;`
Sintaxe	`addi $t, $s, imm`
Codificação	`0010 00ss ssst tttt iiii iiii iiii iiii`

AND – Bitwise AND

Descrição	Faz a conjunção bit a bit de dois registos e armazena o resultado num registo
Operação	`$d = $s & $t;`
Sintaxe	`and $d, $s, $t`
Codificação	`0000 00ss ssst tttt dddd d000 0010 0100`

ANDI – Bitwise AND immediate

Descrição	Faz a conjunção bit a bit de um registo e um valor imediato e armazena o resultado num registo
Operação	`$t = $s & imm;`
Sintaxe	`andi $t, $s, imm`
Codificação	`0011 00ss ssst tttt iiii iiii iiii iiii`

BEQ – Branch on equal

Descrição	Salta se os dois registos são iguais
Operação	`if $s == $t PC += offset << 2;`
Sintaxe	`beq $s, $t, offset`

Codificação	`0001 00ss ssst tttt iiii iiii iiii iiii`

BNE – Branch on not equal

Descrição	Salta se os dois registos não são iguais
Operação	`if $s != $t PC += offset << 2;`
Sintaxe	`bne $s, $t, offset`
Codificação	`0001 01ss ssst tttt iiii iiii iiii iiii`

J – Jump

Descrição	Salta para o endereço calculado.
Operação	`PC = target << 2;`
Sintaxe	`j target`
Codificação	`0000 10ii iiii iiii iiii iiii iiii iiii`

JAL – Jump and link

Descrição	Salta para o endereço calculado e guarda o endereço de retorno em $ra
Operação	`$ra = PC+4; PC = target << 2;`
Sintaxe	`jal target`
Codificação	`0000 11ii iiii iiii iiii iiii iiii iiii`

JR – Jump register

Descrição	Salta para o endereço contido em $s.
Operação	`PC = $s`
Sintaxe	`jr $s`
Codificação	`0000 00ss sss0 0000 0000 0000 0000 1000`

LB – Load byte

Descrição	O byte do endereço especificado é carregado no registo $t
Operação	`$t = MEM[$s + offset]`
Sintaxe	`lb $t, offset($s)`

Codificação	`1000 00ss ssst tttt iiii iiii iiii iiii`

LUI – Load upper immediate

Descrição	O valor imediato é deslocado 16 bits e colocado no registo. Os 16 bits menos significativos são zeros.
Operação	`$t = (imm << 16);`
Sintaxe	`lui $ t, imm`
Codificação	`0011 11-- ---t tttt iiii iiii iiii iiii`

LW – Load word

Descrição	A palavra do endereço especificado é carregado no registo $t
Operação	`$t = MEM[$s + offset]`
Sintaxe	`lw $t, offset($s)`
Codificação	`1000 11ss ssst tttt iiii iiii iiii iiii`

NOOP – no operation

Descrição	Não executa nenhuma operação.
Operação	`PC += 4;`
Sintaxe	`noop`
Codificação	`0000 0000 0000 0000 0000 0000 0000 0000`

Nota: A codificação da `NOOP` representa a instrução `SLL $0,$0,0`, o que não tem efeitos colaterais. Na verdade, quase todas as instruções que têm $0 como registo de destino não têm efeito colaterais e podem, portanto, ser consideradas uma instrução `NOOP`.

OR – Bitwise OR

Descrição	Faz a disjunção bit a bit de dois registos e armazena o resultado num registo
Operação	`$d = $s | $t;`
Sintaxe	`or $d, $s, $t`
Codificação	`0000 00ss ssst tttt dddd d000 0010 0101`

ORI – Bitwise OR immediate

Descrição	Faz a disjunção bit a bit de um registo e um valor imediato e armazena o resultado num registo
Operação	`$t = $s | imm;`
Sintaxe	`ori $t, $s, imm`
Codificação	`0011 01ss ssst tttt iiii iiii iiii iiii`

SB – Store byte

Descrição	O byte menos significativo de $t é guardado no endereço especificado
Operação	`MEM[$s + offset] = (0xff & $t)`
Sintaxe	`sb $t, offset($s)`
Codificação	`1010 00ss ssst tttt iiii iiii iiii iiii`

SLL – Shift left logical

Descrição	Desloca para a esquerda o valor do registo pela quantidade listada na instrução e coloca o resultado num outro registo. São colocados zeros à direita.
Operação	`$d = $t << h;`
Sintaxe	`sll $d, $t, h`
Codificação	`0000 00ss ssst tttt dddd dhhh hh00 0000`

SLLV – Shift left logical variable

Descrição	Desloca para a esquerda o valor do registo pela quantidade listada no segundo registo e coloca o resultado num terceiro registo. São colocados zeros à direita.
Operação	`$d = $t << $s;`
Sintaxe	`sllv $d, $t, $s`
Codificação	`0000 00ss ssst tttt dddd d--- --00 0100`

SLT – Set on less than (signed)

Descrição	Se $s é inferior a $t, $d é colocado a um; caso contrário recebe zero.

Operação	`if $s<$t $d=1; else $d=0;`
Sintaxe	`slt $d, $s, $t`
Codificação	`0000 00ss ssst tttt dddd d000 0010 1010`

SLTI - Set on less than immediate (signed)

Descrição	Se $s é inferior a imm , $d é colocado a um; caso contrário recebe zero.
Operação	`if $s<imm $t=1; else $t=0;`
Sintaxe	`slti $t, $s, imm`
Codificação	`0010 10ss ssst tttt iiii iiii iiii iiii`

SRA – Shift right arithmetic

Descrição	Desloca para a direita o valor do registo pela quantidade listada na instrução e coloca o resultado num outro registo. O bit de sinal é inserido à direita.
Operação	`$d = $t >> h;`
Sintaxe	`sra $d, $t, h`
Codificação	`0000 00-- ---t tttt dddd dhhh hh00 0011`

SRL – Shift right logical

Descrição	Desloca para a direita o valor do registo pela quantidade listada na instrução e coloca o resultado num outro registo. São colocados zeros à esquerda.
Operação	`$d = $t >> h;`
Sintaxe	`srl $d, $t, h`
Codificação	`0000 00-- ---t tttt dddd dhhh hh00 0010`

SRLV - Shift right logical variable

Descrição	Desloca para a direita o valor do registo pela quantidade listada no segundo registo e coloca o resultado num terceiro registo. São colocados zeros à esquerda.
Operação	`$d = $t >> $s;`
Sintaxe	`srlv $d, $t, $s`

Codificação	`0000 00ss ssst tttt dddd d000 0000 0110`

SUB – Subtract

Descrição	Subtrai dois registos e armazena o resultado num registo.
Operação	`$d = $s – $t;`
Sintaxe	`sub $d, $s, $t`
Codificação	`0000 00ss ssst tttt dddd d000 0010 0010`

SW – Store word

Descrição	O conteúdo de $t é guardado no endereço especificado
Operação	`MEM[$s + offset] = $t`
Sintaxe	`sw $t, offset($s)`
Codificação	`1010 11ss ssst tttt iiii iiii iiii iiii`

XOR – Bitwise exclusive or

Descrição	Realiza o OR exclusivo de dois registos e armazena o resultado num registo.
Operação	`$d = $s ^ $t;`
Sintaxe	`xor $d, $s, $t`
Codificação	`0000 00ss ssst tttt dddd d--- --10 0110`

XORI – Bitwise exclusive or immediate

Descrição	Realiza o OR exclusivo entre um registo e um valor e armazena o resultado num registo.
Operação	`$t = $s ^ imm;`
Sintaxe	`xori $t, $s, imm`
Codificação	`0011 10ss ssst tttt iiii iiii iiii iiii`