UNIVERSIDADE DE SÃO PAULO

GUSTAVO POMPERMAYER FULANETTI SILVA
JOÃO GABRIEL DE SENNA LAMOLHA

RELATÓRIO MIPS - ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES 1

SÃO PAULO
2024

GUSTAVO POMPERMAYER FULANETTI SILVA
JOÃO GABRIEL DE SENNA LAMOLHA

RELATÓRIO MIPS - ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES 1

Relatório Final apresentado à disciplina Organização e Arquitetura de Computadores 1 pela Universidade de São Paulo.
Orientadora: Profª. Dra. Gisele da Silva Carneiro.

SÃO PAULO
2024

ÍNDICE

1. ARQUITETURA MIPS

1.1 - Panorama geral

A arquitetura MIPS é uma arquitetura de conjunto de instruções (ISA) que se destaca pela sua simplicidade e eficiência, haja vista que é uma arquitetura RISC (Reduced Instruction Set Computer). Isso significa que ela foi feita para ser mais simples e agradável para os programadores da época. Além disso, também foi amplamente utilizada em CPUs para uma variedade de aplicações, desde dispositivos embarcados até supercomputadores.

Tipos de Dados: Todas as instruções são de 32 bits Byte = 8 bits, Halfword = 2 bytes, Word = 4 bytes. Um caractere ocupa 1 byte na memória, um inteiro ocupa 1 word(4 bytes) na memória.

1.2 - Registradores e seus tipos:

Existem 4 tipos distintos de registradores nesta arquitetura, e são eles os:

Registradores de Dados: Armazenam dados temporários e resultados de operações.

Registradores de Endereços: Contém endereços de memória, usados para acessar dados na memória principal.

Registradores de Status ou de Condição: Mantêm informações sobre o estado de uma operação ou do processador.

Registradores de Controle: Controlam operações específicas dentro do processador.

# do Reg.	Nome	Descrição
0	$zero	retorna o valor 0
1	$at	(assembler temporary) reservado pelo assembler
2~3	$v0-$v1	(values) das expressões de avaliação e resultados de função
4~7	$a0-$a3	(arguments) Primeiros quatro parametros para subrotinas. Não é preservado em chamadas de procedures.
8~15	$t0-$t7	(temporaries) Subrotinas pode usar salvando-os ou não. Não é preservado em chamadas de procedures.
16~23	$s0-$s7	(saved values) Uma subrotina que usa um desses deve salvar o valor original e restaurar antes de terminar. Preservados na chamada de procedures.
24~25	$t8-$t9	(temporaries) Usados em adição aos $t0-$t7
26~27	$k0-$k1	Reservados para uso do tratamento de interrupção/trap.
28	$gp	(global pointer) Aponta para o meio do block de 64k de memoria no segmento de dados estaticos.
29	$sp	(stack pointer) Aponta para o top da pilha
30	$s8/$fp	(saved values/ frame pointer) Preservado na chamada de procedures.
31	$ra	(return address)
	$f0	Recebe o retorno de floats em funções
	$f12/$f14	Usados para passar floats para funções
	($f12,$f13) ($f14,$f15)	Usados em conjunto para passar doubles para funções

Fonte: Dpto de Informática UFPR

Aqui podemos ver claramente vários tipos diferentes de registradores e suas funcionalidades, como quais são reservados, para quais objetivos, o que eles fazem, etc.

1.3 - Tipos de Instrução

Em arquitetura MIPS, os tipos de instruções R, I e J são categorias que agrupam diferentes tipos de instruções baseadas na operação que realizam e no formato de sua representação binária. Deste modo, se pegarmos o caminho de dados, podemos observar claramente as nuances em cada uma delas, além de suas semelhanças. Aqui está uma explicação sucinta de cada uma:

1.3.1 - Instruções R, I, J.

Instruções R:

As instruções do tipo R são também conhecidas como R-R (Registrador - Registrador), são principalmente instruções lógico -aritméticas de 32 bits. Exemplos: add, sub, and, or, slt, falaremos mais adiante sobre as instruções.

Seus 32 bits de instrução são distribuídos da seguinte maneira:

OP	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	**5 bits**	**5 bits**	**5 bits**	**5 bits**	**6 bits**

Campo OP:

O opcode é o campo que especifica a operação que a instrução deve realizar. Em outras palavras, ele indica qual tipo de instrução está sendo executada.

O tamanho típico do opcode em MIPS é de 6 bits.

Exemplos de opcodes em MIPS incluem operações como add, sub, lw (load word), sw (store word), beq (branch if equal), entre outras.

Campo rs, rt, rd:

rs, rt e rd são campos que indicam os registradores envolvidos na operação especificada pelo opcode.

rs (Source Register): Indica o registrador fonte 1, de onde os dados serão lidos.

rt (Target Register): Indica o registrador fonte 2, de onde os dados serão lidos.

rd (Destination Register): Indica o registrador destino, onde o resultado da operação será armazenado.

Campo shamt e funct:

O campo funct é usado para especificar a operação específica a ser realizada na instrução do tipo R.

Ele define a função ou ação que a instrução irá executar dentro da categoria de instruções do tipo R.

No formato da instrução do tipo R em MIPS, os bits no campo funct diferenciam entre várias operações, como adição (add), subtração (sub), AND (and), OR (or), set less than (slt), entre outras.

O campo funct geralmente tem um tamanho de 6 bits, permitindo até 64 códigos de função diferentes (2^6).

Já o campo shamt é um campo de bits usado em instruções de deslocamento (shift) e rotação (rotate) nos processadores MIPS.

Ele especifica a quantidade de bits a serem deslocados à esquerda (shift left) ou à direita (shift right) em operações como sll (shift left logical), srl (shift right logical) e sra (shift right arithmetic).

Normalmente, o shamt é um valor de 5 bits, limitando o deslocamento máximo a 31 bits (2^5 - 1).

Instruções I:

Instruções do tipo I na arquitetura MIPS são aquelas em que os operandos incluem um valor imediato (constante) e um ou mais registradores. Essas instruções são usadas principalmente para operações que envolvem acesso à memória, operações aritméticas com valores imediatos, e comparações condicionais.

Separação de bits:

op	**rs**	**rt**	**constant or adress**
6 bits	**5 bits**	**5 bits**	**16**

Campo constant or address:

Em instruções MIPS, como addi, lw, sw, beq, bne, etc., a "constante" se refere a um valor específico que a instrução usa diretamente.

Exemplo: Na instrução addi $t0, $t1, 10, o número 10 é a constante. Ele é usado como parte da operação de adição.

Endereço (Address):

Quando uma instrução MIPS, como lw (load word) ou sw (store word), precisa acessar a memória principal, o "endereço" é o local exato na memória onde os dados estão armazenados ou para onde serão armazenados.

Exemplo simples: Na instrução lw $t0, 0($t1), 0($t1) representa o endereço. Isso indica onde na memória o processador deve procurar para carregar o valor no registrador $t0.

Esses campos são usados para especificar valores diretos (constantes) em operações aritméticas ou lógicas, e para indicar onde na memória os dados devem ser lidos ou escritos em operações de acesso à memória. Eles são fundamentais para o funcionamento correto e eficiente dos programas escritos.

Instruções do tipo J:

Instruções do tipo J na arquitetura MIPS se classificam como um conjunto de comandos utilizados para alterar o fluxo de execução de um programa, fazendo com que ele dê um salto para outro ponto do código. As instruções do tipo J são úteis para criar estruturas de controle, como loops e funções.

Existem dois principais tipos de instruções do tipo J:

1. J (Jump) - Salta para um endereço em específico.
2. JAL (Jump And Link) - Salta para um endereço específico e armazena o endereço de retorno ao registrador $ra (registrador 31), afim de que o programa possa retornar ao ponto de saída após a execução da sub-rotina.

Separação de bits:

OP	**Endereço**
6 bits	**26 bits**

Exemplo:

Na instrução de tipo J, a função J 0x00400010 faz com que o programa salte diretamente para o endereço 0x00400010, modificando o fluxo de execução do programa para aquele ponto específico do código.

Na instrução de tipo JAL, por sua vez, JAL 0X00400020, o programa salta para o endereço 0x00400020 e ao mesmo tempo armazena o endereço de retorno no registrador $ra (registrador 31). Dessa forma, o programa retornará à instrução seguinte após a conclusão da sub-rotina ou função chamada.

1.3.2 - Aritmética

Add - R	add $1,$2,$3	$1 = $2 + $3 (signed)	Adiciona dois registros, estende sinal de largura de registro.
Add unsigned - R	addu $1,$2,$3	$1 = $2 + $3 (**un**signed)	Soma sem extensão de sinal de largura de registro.
Subtract - R	sub $1,$2,$3	$1 = $2 – $3 (signed)	Subtrai dois registradores e guarda no terceiro
Subtract unsigned - R	subu $1,$2,$3	$1 = $2 – $3	Subtração sem extensão de sinal de largura de registro
Add immediate - I	addi $1,$2,Constante	$1 = $2 + Constante	Usado para adicionar constantes, estende sinal de largura de registro.

Add Immediate unsigned - I	addiu $1,$2,CONST	$1 = $2 + Constante	Para adicionar constantes, mas sem extensão de sinal de largura de registro.
Multiply - R	mult $1,$2	LO = (($1 * $2) << 32) >> 32; HI = ($1 * $2) >> 32;	Multiplica dois registradores e coloca o resultado de 64 bits em dois pontos especiais de memória - LOW e HI. Como alternativa, pode-se dizer que o resultado dessa operação é: (int HI, int LO) = (64-bit) $1 * $2.
Divide - R	$div $1, $t2	LO - Parte inteira Hi = % da divisão	Dois regs são divididos, o resto da divisão é armazenado em hi e a parte inteira em LO

Fonte: adaptado do site (Wikipédia, 2023)

1.3.3 - Instruções condicionais (de desvio e de salto)

beq $1,$2,CONST - I	if ($1 == $2) go to PC+4+CONST	Vai para a instrução no endereço especificado se os dois registradores tiverem o mesmo valor.
bne $1,$2,CONST - I	if ($1 != $2) go to PC+4+CONST	Vai para a instrução no endereço especificado se os dois registradores tiverem valor diferente.

Jump - J	j CONST	Muda o registrador PC(registrador que guarda o endereço da próx instrua ser executada) para o valor do label
Jump register - R	jr $1	Muda o registrador PC(registrador que guarda o endereço da próxima instrução a ser executada) para o endereço contido no registrador
Jump and link - J	jal CONST	

Fonte: adaptado do site (Wikipédia, 2023)

1.3.4 - Lógicas

And - R	and $1,$2,$3	$1 = $2 & $3	Guarda o resultado da operação lógica AND
And immediate - I	andi $1,$2,CONST	$1 = $2 & CONST	Guarda o resultado da operação lógica AND, reg com constante

Or - R	or $1,$2,$3	$1 = $2 | $3	Guarda o resultado da operação logica OR
Or immediate - I	ori $1,$2,CONST	$1 = $2 | CONST	Guarda resultado da operação Or, entre reg e const
Exclusive or - R	xor $1,$2,$3	$1 = $2 ^ $3	Guarda resultado da operação Or, entre reg e const
Nor - R	nor $1,$2,$3	$1 = ~($2 | $3)	Guarda o resultado da operação lógica NOR
Set on less than - R	slt $1,$2,$3	$1 = ($2 < $3)	Verifica se um reg é menor que o outro
Set on less than immediate - I	slti $1,$2,CONST	$1 = ($2 < CONST)	Verifica se o reg é menor que uma const

Fonte: adaptado do site (Wikipédia, 2023)

1.3.5 - Transferência de dados

Load word - I	lw $1,CONST ($2)	$1 = Memory[$2 + CONST]	Carrega 4 bytes da memória a partir de [$s2 + CONST] para $t0.

Load halfword - I	lh $1,CONST ($2)	$1 = Memory[$2 + CONST] (signed)	Usada para carregar uma meia palavra (16 bits) da memória para um registrador..
Load halfword unsigned- I	lhu $1,CONST ($2)	$1 = Memory[$2 + CONST] (unsigned)	Carregar 16 bits (meia palavra) da memória para um registrador, sem estender o sinal.
Load byte - i	lb $1,CONST ($2)	$1 = Memory[$2 + CONST] (signed)	Carrega o byte armazenado
Load byte unsigned- i	lbu $1,CONST ($2)	$1 = Memory[$2 + CONST] (unsigned)	Carrega byte sem extensão de sinal
Store word - i	sw $1,CONST ($2)	Memory[$2 + CONST] = $1	Usada para armazenar uma palavra (32 bits) de um registrador na memória
Store half - i	sh $1,CONST ($2)	Memory[$2 + CONST] = $1	Armazena a primeira metade do um registrador e o byte seguinte.
Store byte - i	sb $1,CONST ($2)	Memory[$2 + CONST] = $1	Armazena o primeiro quarto de um registro (um byte) em

Load upper immediate - i	lui $1,CONST	$1 = CONST << 16	Usada para carregar um valor imediato nos 16 bits superiores de um registrador, deixando os 16 bits inferiores como zeros.
Move from high - R	mfhi $1	$1 = HI	Move um valor de HI a um registrador.
Move from low - R	mflo $1	$1 = LO	Move um valor de LO a um registro.
Move from Control Register - R	mfcZ $1, $2	$1 = Coprocessor[Z].ControlRegister[$2]	Utilizada para mover o conteúdo de um registrador de controle específico para um registrador geral.
Move to Control Register - R	mtcZ $1, $2	Coprocessor[Z].ControlRegister[$2] = $1	Utilizada para mover o conteúdo de um registrador geral para um registrador de controle específico.
Load word coprocessor - I	lwcZ $1,CONST ($2)	Coprocessor[Z].DataRegister[$1] = Memory[$2 + CONST]	É usada para carregar uma palavra (32 bits) da memória para um registrador de ponto flutuante (coprocessador 1).
Store word coprocessor - I	swcZ $1,CONST ($2)	Memory[$2 + CONST] = Coprocessor[Z].DataRegister[$1]	É usada para armazenar uma palavra (32 bits) de um

			registrador do CPC1 para a memória

Fonte: adaptado do site (Wikipédia, 2023)

2. Instruções detalhadas utilizadas no programa

Microoperações:

Todo programa contém um conjunto de instruções e seus opcodes que ditam o que vai ser executado. O fato é que todos os programas podem ser decompostos em operações elementares menores, a fim de facilitar a compreensão e o entendimento profundo do que está acontecendo. Deste modo, as microoperações aqui descritas nos ajudarão a trazer clareza em relação ao código aplicado para resolução do problema em questão.

LI - Load Immediate - PARAMETROS: $REG, CONST: Formato I

Recebe uma constante como parametro e transfere para o registrador indicado. No IR, é atribuído o opcode soma, o endereço do registrador, e o valor imediato, além do registrador $zero.

ADDI - Formato I - Parâmetros $REG1, CONST:

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Salva o resultado da soma do reg1 com $zero e com o imediato.

ADD - Formato R - Parâmetros $REG1, $REG, $REG:

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. O valor da operação entre ambos os registradores é armazenado em $REG1.

LW - LOAD WORD - TIPO I - Parâmetros $REG1, Variável

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. O valor é trazido da memória para o $REG1

LWc1 - LOAD WORD - TIPO I - Parâmetros $Float1, Variável

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. O valor é trazido da memória para o registrador do tipo $Float.

JAL - Jump And Link - TIPO J - Parametros: JAL LABEL

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Salta diretamente para a label, uisando extensor de sinal.

SW - SAVE WORD - TIPO I I Parâmetros $REG1, Variável

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Salva a palavra lida na memória.

LA - LOAD ADRESS - TIPO I - $REG1, $REG3.

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. O IR é colocado no OPCODE da soma. Desta forma, é calculado o endereço a ser carregado.

MUL - MULTIPLICAÇÃO - TIPO R- Paramêtros: $REG1 $REG2 $REG3.

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Multiplica o registrador1 pelo registrador2 e guarda no registrador0.

JR $RA - Jump Return - TIPO J - Parametros: jr $ra

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Desvia para o endereço contido em RA - usado para fazer o retorno da subrotina para o programa.

MOVE - MOVER - TIPO I - Parametros: registrador0, registrador 1.

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Copia o valor do registrador1 para o registrador0.

MOV.S - MOVER SINGLE POINT - TIPO I - Parametros: $f1, $f2

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Copia o valor de um registrador tipo float para outro registrador do tipo float.

BGT - BRANCH IF GREATER THAN - TIPO I - Parametros: $r1,$r2, label

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Se a for verdade, faz um salto para a label.

c.lt.s - COMPARE IF LESS THAN SINGLE PRECISION - Parametros: $f1,$f2.

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Compara floats, se for menor, retorna um valor 0.

BC1T - BRANCH IF CONDITION FLAG IS TRUE - TIPO I - Parametros: bc1t label

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. Se o valor retornado for 0 (true), pula para a label indicada

Syscall - chamada do sistema - Parametros: li v0, const;

Busca pela instrução e pelo operando se inicia. Após encontrada, é feita a operação. O sistema operacional toma uma ação baseado no código inserido no registrador $v0.

3. Código em alto nível e explicação.

20 (a) - Dado um vetor real X com n elementos e um certo índice k, escreva uma função que determina o índice do elemento mínimo entre X[k], X[k+1], ..., X[n-1].

O código foi implementado na linguagem de programação C e compilado no Replit e CodeBlocks em suas versões mais recentes. A lógica aplicada foi: escolher o menor valor sendo o mesmo valor de V[k], pois como iríamos percorrer o vetor a

partir da posição K, caso não houvesse um valor menor, apenas seria retornado o índice K mesmo.

```
#include <stdio.h>
#include <stdlib.h>

int encontraMenorValor(){
float v[] = {1.01,2.24,3.36,0.9,6.26,0.95};
int n = 6;
int k = 1;
int i;
float menorValor = v[k];
int menorIndice = k;

 for(i = k; i<=n; i++){
    if(menorValor > v[i]) {
    menorValor = v[i];
    }
    else{
        continue;
    }
  menorIndice = i;
}
return menorIndice;
}

int main()
{
     printf(" Indice: %i\n", encontraMenorValor());
   return 0;
}
```

4. Código em MIPS

```
.data
        v: .float 1.01, 2.24, 3.36, 0.91, 0.26, 0.0  # Vetor de floats
        tam: .word 6                    # Número de elementos no vetor v
        k: .word 1                      # Posição inicial k, pode ser ajustada
conforme o necessário
        retorno: .word 0
        espaco: .asciiz "\n Indice "
.text
        main:
        la $a0, v                   # carrego o endereço do vetor
        lw $t1, tam                 # tam do vetor
        lw $t2, k                   # valor do indice k
        add $t0, $t2, $zero         #inicio do contador em K
        li $t3, 4                    # nmro de bits de cada posição do vetor
        mul $t4, $t2, $t3            #calcula deslocamento
        add $a1, $a0, $t4            # indice k, resultado
        lwc1 $f1, 0($a1)             # conteúdo do indice K

        addi $sp, $sp, -16           # reserva espaço para 4 palavras (4 * 4 bytes)
        sw $ra, 12($sp)              # salva o registrador de retorno (link) na pilha
      sw $t0, 8($sp)                # salva $t0 na pilha
        sw $a1, 4($sp)               # salva $a1 na pilha
        sw $t1, 0($sp)               # salva $t1 na pilha

        jal loop_compare

        lw $t1, 0($sp)              # restaura $t1
        lw $a1, 4($sp)              # restaura $a1
        lw $t0, 8($sp)               # restaura $t0
        lw $ra, 12($sp)             # restaura $ra
        addi $sp, $sp, 16           # libera o espaço alocado na pilha
```

```
        li $v0, 10
        syscall

        update:
        mov.s $f1, $f3
        addi $t5, $t0, 0
        bgt, $t1, $t0, loop_compare

        loop_compare:
        lwc1 $f3, 0($a1)            #carrega o valor float de k+1
        addi $a1,$a1, 4              # passo p proxima posição do vetor)
        addi $t0, $t0, 1             # acrescento o contador
        c.lt.s $f3, $f1              #comparo, se a condição for vdd, vou para a
sub rotina sem_att, que verifica se o vetor cehegou ao fim, se chegou, imprime
e volta pra o jal, encerrando

        bc1t update             # atualiza o valor se a condição retornar verdadeira

        bgt, $t1, $t0, loop_compare
        # compara se o contador não passou o tamanho max do vetor



#impressão
        print_min_value:
        li $v0, 4
        la $a0, espaco
        syscall

        li $v0, 1
        move $a0, $t5
        syscall

        jr $ra
```

REFERÊNCIAS

ARQUITETURA MIPS. *in*: WIKIPÉDIA: a enciclopédia livre. [São Francisco, CA: Fundação Wikimedia], 2023. Disponível em: https://pt.wikipedia.org/wiki/Arquitetura_MIPS. Acesso em: 27 jun. 2024.

PATTERSON, D.; HENNESSY, J. **Organização e Projeto de Computadores:** a Interface Hardware/Software 5. ed. Rio de Janeiro: Elsevier, 2017. Disponível em: https://pergamum.uel.br/acervo/161399/referencia. Acesso em: 27 jun. 2024.

STALLINGS, W. **Arquiteturas e Organização de Computadores**. 8. ed. São Paulo: Pearson Education Brasil, p. 624, 2010. Disponível em: https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=http://www.telecom.uff.br/orgarqcomp/arq/arquitetura-e-organizacao-computadores-8a.pdf&ved=2ahUKEwic6oiQxoGHAxVqppUCHarbDgYQFnoECB4QAQ&usg=AOvVaw2Up5mvrl5nPnY62kPR7E76. Acesso em: 27 jun. 2024.

Guia Rápido MIPS Tipos de Dados e Formatações. [s.l: s.n.]. Disponível em:

<https://www.inf.ufpr.br/wagner/ci243/GuiaMIPS.pdf>. Acesso em: 27 jun. 2024