UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA -
ESCOLA SUPERIOR DE TECNOLOGIA - EST

JACKSON KELVIN DE SOUZA

Conjunto de instruções do processador MIPS

MANAUS-AM

2016

JACKSON KELVIN DE SOUZA

PROFESSOR: CARLOS MAURICIO S. FIGUEIREDO

MATERIA: ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

MANAUS - AM

2016

Introdução

O MIPS é o nome de uma arquitetura de processadores baseados no uso de registradores. As suas instruções têm à disposição um conjunto de 32 registradores para realizar as operações. Entretanto, alguns destes registradores não podem ser usados por programadores, pois são usados pela própria máquina para armazenar informações úteis.

Processadores MIPS são do tipo RISC (Reduced Instruction Set Computer - ou seja, Computadores com Conjunto de Instruções Reduzidas). Isso significa que existe um conjunto bastante pequeno de instruções que o processador sabe fazer. Combinando este pequeno número, podemos criar todas as demais operações.

Pela sua elegância e simplicidade, processadores MIPS são bastante usados em cursos de arquiteturas de muitas universidades. Ele é considerado um processador bastante didático.

Projetos MIPS são atualmente bastante usados em muitos sistemas embarcados como dispositivos Windows CE, roteadores Cisco e video-games como o Nintendo 64, Playstation, Playstation 2 e Playstation Portable.

Arquitetura MIPS

O MIPS é uma arquitetura baseada registrador, ou seja, a CPU usa apenas registradores para realizar as suas operações aritméticas e lógicas. Processadores baseados no conjunto de instruções do MIPS estão em produção desde 1988. Ao longo do tempo foram feitas várias melhorias do conjunto de instruções. As diferentes revisões que foram introduzidas são MIPS I, MIPS II, MIPS III, MIPS IV e MIPS V. Cada revisão é um super conjunto de seus antecessores. Quando a MIPS Technologies saiu da Silicon Graphics em 1998, a definição da arquitetura foi alterada para definir um conjunto de instruções MIPS32 de 32 bits e um MIPS64 de 64 bits.

Existem diversas extensões opcionais, incluindo MIPS-3D, que é um conjunto simples de instruções SIMD de ponto flutuante, dedicado a tarefas comuns em 3D; MDMX (MaDMaX) que é um conjunto de instruções SIMD inteiras mais extenso usando os registradores de ponto flutuante de 64 bits; MIPS16e que adicionou compressão ao fluxo de instruções de modo que os programas ocupassem menos espaço na memória; e MIPS MT, que adicionou capacidade multithreading ao MIPS.

Representação das Instruções

Sabemos que tudo dentro do computador é representado por meio de bits - que podem ter o valor de 0s e 1s. Até mesmo as instruções de um processador MIPS. Vamos estudar agora como uma instrução é representada a nível de bits. O MIPS reconhece 3 famílias diferentes de instruções. São elas:

As Instruções Tipo R

Como exemplo de instruções do tipo R, temos: add, addu, sub, subu, or e and.

As Instruções Tipo I

Como exemplo deste tipo de instrução, podemos citar todas aquelas que contam com um valor imediato, como: addi, subi, ori, beq e bnq.

As Instruções Tipo J

Como exemplo deste tipo de instrução, podemos citar: j (jump).

Conjunto de Instruções do MIPS

Agora começamos a estudar as instruções que existem em um computador MIPS. Para testar as instruções caso não tenha uma máquina MIPS, use um simulador como o MARS (MIPS Assembler and Runtime Simulator). O download do simulador MARS pode ser feito no link: http://courses.missouristate.edu/KenVollmar/mars/download.htm.

Instruções Aritméticas

Não existem instruções mais complexas como potências ou raízes quadradas. Somente as operações matemáticas mais simples são representadas.

add $r1, $r2, $r3

Esta instrução soma o conteúdo dos registradores $r2 e $r3 colocando o conteúdo no registrador $r1.

addi $r4, $r1, 9

Agora estamos somando o conteúdo do registrador $r1 com o valor imediato 9 e armazenando o resultado em $r4, o número imediato deve ter 16 bits.

addu $r5, $r6, $r4

Quase igual ao add. Mas agora assumimos que todos os valores são não-negativos.

addiu $r7, $r8, 10

Somamos o conteúdo de $r8 com o valor imediato 10 e armazenamos o resultado em $r7. Assume-se que todos os valores são não-negativos.

sub $r1, $r2, $r3

Subtrai-se o conteúdo de $r3 do conteúdo de $r2 e coloca-se em $r1. Também existe subi, subu e subiu que tem comportamento semelhante a addi, addu e addiu, mas para a subtração.

mult $1,$2

signifiacado:
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.

div $1, $2

Significado:
LO = $1 / $2
HI = $1 % $2

Divide dois registradores e coloca o resultado inteiro de 32 bits em LO e o restante em HI.

Instruções Lógicas

and $r1, $r2, $r3

Realiza uma operação AND bit-a-bit entre $r3 e $r2. O resultado é armazenado em $r1.

andi $r1, $r2, 42

Realiza uma operação AND bit-a-bit entre $r2 e o valor imediato 42. O resultado é armazenado em $r1. O número imediato deve caber em 16 bits.

or $r1, $r2, $r3

Realiza uma operação OR bit-a-bit entre $r3 e $r2. O resultado é armazenado em $r1.

ori $r1, $r2, 42

Realiza uma operação OR bit-a-bit entre $r2 e o valor imediato 42. O resultado é armazenado em $r1. O número imediato deve caber em 16 bits.

xor $r1,$r2,$r3

Realiza uma operação OR Exclusivo bit-a-bit entre $r3 e $r2. O resultado é armazenado em $r1.

nor $r1,$r2,$r3

Realiza uma operação NOT bit-a-bit entre $r3 e $r2. O resultado é armazenado em $r1.

slt $r1,$r2,$r3

Ela armazena 1 em $r1 se $r2 < $r3 e 0 caso contrário.

slti $r1,$r2,CONST

Ela armazena 1 em $r1 se $r2 < CONST e 0 caso contrário.

Instruções de Transferência de Dados

Existem três tipos de instruções capazes de copiar dados da memória para os registradores, são elas:

lw $r1, 4($r2)

Load Word: Esta instrução carrega uma palavra (estrutura de 4 bytes) localizada no endereço representado pela soma do valor armazenado no registrador $r2 mais 4. O resultado é armazenado em $r1.

lh $r1, 6($r3)

Load Half: Esta instrução carrega uma estrutura de 2 bytes localizada no endereço representado pela soma do valor armazenado no registrador $r3 mais o número 6. O resultado é armazenado em $r1.

lb $r1, 16($r2)

Load Byte: Esta instrução carrega um byte (8 bits) localizado no endereço representado pela soma do valor armazenado em $r2 mais o número 16. O resultado é armazenado em $r1.
Para armazenarmos conteúdo de um registrador na memória também existem três comandos:

sw $r1, 4($r2)

Store Word: Esta instrução carrega uma palavra (estrutura de 4 bytes) localizada no registrador $r1 e armazena no endereço representado pela soma do valor armazenado no registrador $r2 mais 4.

sh $r1, 4($r2)

Store Half: Esta instrução carrega uma estrutura de 2 bits localizada no registrador $r1 e armazena no endereço representado pela soma do valor armazenado no registrador $r2 mais 4.

sb $r1, 4($r2)

Store Byte: Esta instrução carrega um byte (8 bits) localizado no registrador $r1 e armazena no endereço representado pela soma do valor armazenado no registrador $r2 mais 4.

Instruções de desvio condicional

Agora veremos instruções de desvio condicional. A primeira delas é a instrução beq, ou Branch if Equal:

beq $r1, $r2, DESTINO

Vai para a instrução no endereço especificado se dois registradores são iguais.

Além do beq, temos também o bne, ou Branch if Not Equal:

bne $r1, $r2, DESTINO

Vai para a instrução no endereço especificado se dois registradores não são iguais.

Instruções de salto incondicional
j FINAL

Salta incondicionalmente para a instrução no endereço especificado FINAL.

jr $1

Salta para o endereço contido no registrador específico.

As Pseudo-Instruções

A linguagem Assembly de uma máquina costuma ser um reflexo direto de como são implementadas as instruções de um determinado processador. Entretanto, nem todas as instruções que temos à disposição quando programamos em Assembly são instruções verdadeiras para o processador. Algumas delas são na verdade pseudo-instruções.

Pseudo-instruções costumam ser substituídas pelo montador ao gerar instruções para o computador na forma de Lingüagem de Máquina. Pseudo-Instruções são na verdade combinações de mais de uma instrução.

Como exemplo, vamos converter para Assembly o seguinte algoritmo em Python:

```
a=int(input("digite o primeiro numero:"))
b=int(input("digite o primeiro numero:"))
c=a*b
print("resultado:",c)
```

O algoritmo no Assembly do MIPS ficaria assim:

```
.data
        Numero1: .asciiz "informe o primeiro numero: "
        Numero2: .asciiz "informe o segundo numero: "
        Resultado: .asciiz "resulado: "
.text
        li $v0,4
        la $a0,Numero1
        syscall

        li $v0,5
```

```
syscall

move $t0,$v0

li $v0,4
la $a0,Numero2
syscall

li $v0,5
syscall

move $t1,$v0

mult $t0,$t1

mflo $a3

li $v0,4
la $a0,Resultado
syscall

li $v0,1
move $a0,$a3
syscall
```

Conclusão

Assembly ou linguagem de montagem é uma notação legível por humanos para o código de máquina que uma arquitetura de computador específica usa, utilizada para programar dispositivos computacionais, como microprocessadores e microcontroladores.

Cada arquitetura de computador tem a sua própria linguagem de máquina e, portanto, a sua própria linguagem de montagem. Essas linguagens de montagem diferem no número e tipo de operações que suportam. Também têm diferentes tamanhos e números de registradores, e diferentes representações dos tipos de dados armazenados. Enquanto todos os computadores de utilização genérica são capazes de desempenhar essencialmente as mesmas funções, o modo como o fazem é diferente.

Quando o MIPS estava sendo desenvolvido, quatro regras foram definidas para guiar o projeto. Elas são a filosofia do MIPS:

A simplicidade favorece a regularidade.

O menor é (quase sempre) mais rápido. Levando em conta que uma corrente elétrica se propaga cerca de um palmo por nanosegundo, circuitos simples são menores e portanto, mais rápidos.

Um bom projeto demanda compromissos.

O caso comum deve ser mais rápido. É muito melhor tornar uma instrução que é usada 90% do tempo 10% mais rápida do que fazer com que uma instrução usada em 10% das vezes torne-se 90% mais rápida.

Foram estas as regras usadas para decidir vários aspectos do funcionamento do MIPS - incluindo quais seriam suas instruções e como elas funcionariam.

Referencias

Arquitetura MIPS:
Disponível em: < https://pt.wikipedia.org/wiki/Arquitetura_MIPS >. Acesso em: 31/05/2016 às 17h00min.

O que é o MIPS?
Disponível em: < https://pt.wikibooks.org/wiki/Introdu%C3%A7%C3%A3o_%C3%A0_Arquitetura_de_Computadores/O_que_%C3%A9_o_MIPS%3F >. Acesso em 31/05/2016 às 17h10min.

Instruções do MIPS:
Disponível em: < https://pt.wikibooks.org/wiki/Introdu%C3%A7%C3%A3o_%C3%A0_Arquitetura_de_Computadores/Instru%C3%A7%C3%B5es_do_MIPS >. Acesso em 31/05/2016 às 17h20min.

Representação das Instruções:
Disponível em: < https://pt.wikibooks.org/wiki/Introdu%C3%A7%C3%A3o_%C3%A0_Arquitetura_de_Computadores/Representa%C3%A7%C3%A3o_das_Instru%C3%A7%C3%B5es >. Acesso em 31/05/2016 às 17h30min.

Assembly:
Disponível em: < https://pt.wikipedia.org/wiki/Assembly >. Acesso em 31/05/2016 às 17h40min.

AssemblyMIPS.pdf
Disponível em: < www.*gec.di.uminho.pt/lesi/ac20203/AssemblyMIPS.pdf* >. Acesso em 01/06/2016 às 16h40min.

L*abAssembly.pdf*
Disponível em: < www.inf.ufpr.br/roberto/ci210/assembly/labAssembly.pdf >. Acesso em 01/06/2016 às 16h50min.

Introdução ao Assembly usando o Simulador SPIM
Disponível em: < www.cee.uma.pt/people/faculty/pedro.campos/docs/guia-AC.pdf >. Acesso em 01/06/2016 às 17h00min.