Arquiteturas de Computador II

EPC2 – Arquitetura MIPS

Rodrigo Rufino
Márcio Rotella
Pedro Moreira

ÍNDICE

1. ARQUITETURA DE CONJUNTO DE INSTRUÇÕES MIPS

O MIPS é uma arquitetura RISC simples, simplificada e altamente escalável que está disponível para licenciamento. Ao longo do tempo, a arquitetura evoluiu, adquiriu novas tecnologias e desenvolveu um ecossistema robusto e um apoio abrangente da indústria. Suas características fundamentais - como o grande número de registradores, o número e o caráter das instruções, e os slots de atraso de pipeline visíveis - permitem que a arquitetura MIPS ofereça o maior desempenho por milímetro quadrado para núcleos IP licenciáveis, bem como níveis altos De eficiência de energia para os projetos SoC atuais.

A arquitetura MIPS é uma das mais amplamente suportadas de todas as arquiteturas de processadores, com uma ampla infra-estrutura de ferramentas, software e serviços padrão para ajudar a garantir um desenvolvimento rápido, confiável e econômico. Os desenvolvedores de microprocessadores que querem a máxima flexibilidade do IP do processador têm uma solução na arquitetura MIPS.

Os produtos MIPS Architecture incluem

As arquiteturas de instruções MIPS32 e MIPS64, que são perfeitamente compatíveis, permitem que os clientes migrem de uma geração para a outra, preservando ao mesmo tempo o investimento em software existente.

MicroMIPS, uma arquitetura de instrução (ISA) de compactação de código composta de instruções de 16 e 32 bits, que fornece desempenho semelhante ao MIPS32 com uma redução de tamanho de código de até 35%.

Arquitetura módulos que são abrangidos como parte da arquitetura de base, incluindo SIMD (Single Instruction Multiple Data operação), Virtualização, multi-threading (MT) e tecnologias DSP.

Multi-threading

Pode fazer aparecer um único núcleo de processador e funcionar como vários núcleos separados para melhorar o desempenho e a eficiência.

Virtualização

O módulo de virtualização fornece recursos de segurança aprimorados e suporte para vários sistemas operacionais.

Tecnologia DSP

Apoiar o crescente número de produtos de consumo que exigem uma quantidade crescente de sinal e processamento de mídia cavalo-vapor (cavalo vapor ou cv é uma unidade de medida de potência).

SIMD

SIMD (Single Instruction Multiple Data) melhora o desempenho ao permitir o processamento paralelo eficiente de operações de vetor.

2. PRIMEIRA INSTRUÇÃO MIPS

O MIPS R2000 / R3000 ISA tem instruções de 32 bits de largura fixa. As instruções de largura fixa são comuns para os processadores RISC porque tornam mais fácil buscar instruções sem ter que decodificar. Estas instruções devem ser armazenadas em endereços alinhados por palavras (ou seja, endereços divisíveis por 4).

As instruções MIPS ISA se dividem em três categorias: tipo R, tipo I e tipo J. Nem todos os ISAs dividem suas instruções com clareza. Esta é uma razão para estudar MIPS como uma linguagem assembly primeiro. O formato é simples.

2.1.1 Tipo R

As instruções do tipo R referem-se a instruções do tipo de registro. Dos três formatos, o tipo R é o mais complexo.

Este é o formato da instrução de tipo R, quando é codificado em código de máquina.

B 31-26	B 25-21	B 20-16	B 15-11	B 10-6	B 5-0
Opcode	Registos	Registrar	Registrar	Quantidade de turno	função

A instrução do tipo R é:

Adicione $ rd, $ rs, $ rt

Onde $ rd se refere a algum registro d (d é mostrado como uma variável, no entanto, para usar a instrução, você deve colocar um número entre 0 e 31, inclusive para d). $ Rs, $ rt também são registros.

A semântica da instrução é;

R [d] = R [s] + R [t]

Onde a adição é assinada adição.

Você notará que a ordem dos registros na instrução é o registro de destino ($ rd), seguido dos dois registros de origem ($ rs e $ rt).

No entanto, o formato binário real (mostrado na tabela acima) armazena os dois registos de origem em primeiro lugar, em seguida, o registo de destino. Assim, como o programador de linguagem assembly usa a instrução e como a instrução é armazenada em binário, nem sempre tem que corresponder.

Vamos explicar cada um dos campos da instrução de tipo R.

- Opcode (B_{31-26})

Opcode é abreviação de "código de operação". O opcode é uma codificação binária para a instrução. Opcodes são vistos em todos os ISAs. No MIPS, há um opcode para adicionar.

O opcode em MIPS ISA é apenas 6 bits. Normalmente, isto significa que existem apenas 64 instruções possíveis. Mesmo para um RISC ISA, que normalmente tem poucas instruções, 64 é bastante pequeno. Para as instruções do tipo R, são utilizados 6 bits adicionais (B_{5-0}) chamados função . Assim, os 6 bits do opcode e os 6 bits da função especificam o tipo de instrução para as instruções do tipo R.

- Rd (B_{25-21})

Este é o registro de destino . O registrador de destino é o registrador onde o resultado da operação é armazenado.

- Rs (B_{20-16})

Este é o primeiro registro de origem . O registrador de origem é o registrador que contém um dos argumentos da operação.

- Rt (B_{15-11})

Este é o segundo registro de origem .

- (B_{10-6})

A quantidade de bits a mudar. Usado em instruções de mudança.

- Função (B_{5-0})

Um 6 bits adicionais usados para especificar a operação, além do opcode.

2.1.2 Instruções de tipo I

I-type é abreviação de "tipo imediato". O formato de uma instrução do tipo I se parece com:

B_{31-26}	B_{25-21}	B_{20-16}	B_{15-0}
Opcode	Registos	Registrar	imediato

A instrução do tipo I é vista como:

Adicione $ rt, $ rs, immed

Neste caso, $ rt é o registrador de destino, e $ rs é o único registro de origem. É invulgar que $ rd não seja usado, e que $ rd não apareça nas posições de bit B_{25-21} para instruções de tipo R e I. Presumivelmente, os projetistas do ISA MIPS tiveram suas razões para não fazer o registro de destino em um local específico para tipo R e tipo I.

A semântica da instrução addi é:

$R[t] = R[s] + (IR15)^{16}\ IR_{15-0}$

Onde IR refere-se ao registo de instruções, o registo onde a instrução actual é armazenada. $(IR_{15})^{16}$ significa que o bit B15 do registo de instruções (que é o bit de sinal do valor imediato) é repetido 16 vezes. Isto é então seguido por IR_{15-0}, que é o 16 bits do valor imediato.

Basicamente, a semântica diz para assinar-estender o valor imediato para 32 bits, adicioná-lo (usando adição assinada) para registrar R [s] e armazenar o resultado no registro $ rt .

2.1.3 Instruções de tipo J

Tipo J é abreviação de "tipo de salto". O formato de uma instrução do tipo J se parece com:

B 31-26	B 25-0
Opcode	alvo

A instrução do tipo I é vista como:

J target

A semântica da instrução j (j significa salto) são:

PC <- PC 31-28 IR 25-0 00

Onde PC é o contador do programa, que armazena o endereço atual da instrução que está sendo executada. Você atualiza o PC usando os 4 bits superiores do contador do programa, seguidos pelos 26 bits do alvo (que são os 26 bits mais baixos do registro de instruções), seguido de dois 0's, o que cria um endereço de 32 bits. A instrução de salto será explicada em mais detalhes em um futuro conjunto de notas.

2.1.4 Por que cinco?

Se você observar as instruções do tipo R e I , você verá 5 bits reservados para cada registro. Você pode se perguntar por quê.

MIPS suporta 32 registos inteiros. Para especificar cada registro, o registro é identificado com um número de 0 a 31. Demora $\log_2 32 = 5$ bits para especificar um de 32 registradores.

Se o MIPS tiver registro 64, você precisaria de 6 bits para especificar o registro.

O número de registro é especificado usando um binário não assinado. Assim, 00000 refere-se a $ r0 e 11111 refere-se ao registo $ r31 .

2.1.5 Por que estudar Formatos de Instrução

Você pode se perguntar por que é importante estudar formatos de instruções. Eles parecem ser construídos arbitrariamente. No entanto, eles não são. Por exemplo, é bastante útil ter o opcode do mesmo tamanho e da mesma localização. É útil saber os bits exatos usados para o valor imediato. Isso torna a decodificação muito mais rápida e o hardware para lidar com a descodificação de instruções muito mais simples.

Além disso, você começa a perceber que informações as instruções armazenam. Por exemplo, não é tão óbvio que valores imediatos são armazenados como parte da instrução para instruções de tipo I.

Se você sabe que, por exemplo, addi faz adição assinada, então você também pode concluir que o valor imediato é representado em 2C. Além disso, adicionar o valor imediato a um valor de registro de 32 bits significaria estender o valor imediato a 32 bits.

No entanto, nem todas as instruções de tipo I codificam o 16 bit imediato em 2C. Por exemplo, addiu (adicionar imediata unsigned) interpreta os 16 bits como UB. Ele zero-estende o imediato e, em seguida, adiciona-o ao valor armazenado em um registro de 32 bits.

2.1.6 Três Operandos

Além disso, observe que as instruções de tipo R usam três operandos (ou seja, argumentos). Em ISAs pré-RISC anteriores, a memória era cara, então os projetistas ISA tentaram minimizar o número de bits usados em uma instrução. Isso significava que frequentemente havia dois, um ou nenhum operando.

Como conseguiram isso? Aqui está um exemplo de uma instrução

```
Cisc_add $ r1, $ r2 # R [1] = R [1] + R [2]
```

Uma maneira de reduzir o número total de operandos é fazer com que um operando seja um registrador de origem e um de destino.

Outra abordagem é usar um registro implícito.

```
Acc_add $ r2 # Acc = Acc + R [2]
```

Por exemplo, pode haver um registro especial chamado acumulador . Este registo não é mencionado explicitamente na instrução. Em vez disso, ele está implícito pelo opcode.

Primeiros computadores pessoais, como o Apple 2, ISAs usados com 1 ou 2 registros, e esses registros eram muitas vezes parte da maioria das instruções, assim, eles não precisavam ser especificados.

Com a memória se tornando mais barata e o acesso à memória se tornando mais barato, torna-se mais fácil dedicar mais bits a uma instrução e especificar três operandos em vez de dois. Isso torna mais conveniente para o programador de linguagem de montagem.

3. COMPILAÇÃO DE EXPRESSÕES NO MIPS

Um programa pode, eventualmente, apresentar expressões um pouco mais complexas, envolvendo mais de uma operação com certa ordem lógica de resolução dela. Um exemplo disso são expressões matemáticas de soma e subtração utilizando parênteses.

Como exemplo, a seguinte expressão pode ser tomada:

f = e - (a - b) + (b - c)

Se fosse para ser resolvida normalmente na mão, respeitando os parênteses como em uma expressão matemática qualquer, a ordem seria a seguinte:

f = e - (a - b) + (b - c)

f = e – T1 + T2

f = e – T1 + T2

f = T3 + T2

f = T3 + T2

O código MIPS da expressão ficaria o seguinte (considerando os registradores pra cada variável: a = $s0, b = $s1, c = $s2, d = $s3, e = $s4, f = $s5):

```
SUB   $t1, $ s0, $ s1
SUB   $t2, $ s1, $ s2
SUB   $t3, $ s4, $t1
ADD   $ s5, $t3, $t2
```

Representação das instruções:

Op	Rs	Rt	Rd	Shamt	Funct
000000	16 ou 010 000	17 ou 010 001	9 ou 001 001	00000	100 010
000000	17 ou 010 001	18 ou 010 010	10 ou 001 010	00000	100 010
000000	20 ou 010 100	9 ou 001 001	11 ou 001 011	00000	100 010
000000	11 ou 001 011	10 ou 001 010	21 ou 010 101	00000	100 000

Então, o código de máquina fica da seguinte forma:

00000001000001000100100100000100010
00000001000101001000101000000100010
00000001010000100100101100000100010
00000000101100101001010100000100000

4. CONVERTENDO UMA INSTRUÇÃO COM ARRAY NO MIPS

Para iniciar uma explicação de como acontece a conversão de uma instrução com Array no MIPS, precisa-se conhecer o conceito de instruções de formato Tipo I e load word (LW).

A instruções de formato Tipo I são classificadas como instrução de transferência de dados, possuem 32 bits, porém têm menos campos que a Tipo R (aritméticas).

QUADROS	OPCODE	RS	RT	ENDEREÇO
BITS	6	5	5	16
FUNÇÃO	CÓDIGO DE OP.	Registrador Destino	Registrador Fonte	Endereço de Memória

Representação de instruções do formato de Tipo 1

A LW tem como função transferir dados da memória para os registradores e sempre deve ser utilizada quando se trata de uma Array. Sua sintaxe é:

LW registrador_destino, valor (registrador_fonte)

Assim, supondo a operação a = b + c[10]

O primeiramente deve-se 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 precisa-se 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
```

5. ARMAZENANTO UM VALOR EM ARRAY NO MIPS

Para armazenar uma instrução numa Array, precisa-se conhecer a instrução Store Word(SW).

A SW tem como objetivo armazenar um valor, que está localizado no registrador, na memória. Ela é bem parecida com a LOAD WORD, o que inverte é a fonte com o destino:

SW registrador_fonte, valor (registrador_destino)

Assim, supondo a operação a[15] = b + c

Atribuindo $s0 para a, $s1 para b e $s2 para c. O primeiramente deve-se converter

b + c:

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

A instrução acima realiza uma soma entre b e 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 )
```