Aritmética computacional - Aula 6

Nicolas Chagas Souza

08/08/2022

Arredondamento

Na arquitetura MIPS define-se 4 tipos de arredondamento:

- Sempre para cima:
  - $2,11 \rightarrow 2,2$
  - $2,15 \rightarrow 2,2$
- Sempre para baixo:
  - $2,11 \rightarrow 2,1$
  - $2,15 \rightarrow 2,1$
- Truncamento:
  - $1,01101_2 \rightarrow 1,40625_{10}$
  - $1,011_2 \rightarrow 1,3725_{10}$
- Para o mais próximo:
  - $2,11 \rightarrow 2,1$
  - $2,19 \rightarrow 2,2$
  - Um critério para desempate quando o algarismo menos significativo é 5, arredonda-se para o número par mais próximo:
    - $2,15 \rightarrow 2,2$
    - $2,25 \rightarrow 2,2$

Instruções no MIPS

O processador MIPS tem três coprocessadores, sendo o co-processador 1 reservado para o armazenamento de números em ponto flutuante. Este possui 32 registradores de uso geral: $f0 a $f31.

Não existem instruções imediatas para lidar com números em ponto flutuante, além disso, não é possível realizar operações entre registradores inteiros e reais utilizando as instruções MIPS, para tal é necessário armazenar o número inteiro em um registrador real, e então fazer a operação.

Para precisão dupla (double) utiliza-se os registradores de número par, por exemplo, usando o registrador $f2 em uma instrução, esta avaliará os valores armazenados em $f2 e $f3.

Para carregar constantes utilizamos a seção `.data`:

- `float f1, f2, f3, ..., fn`
- `double d1, d2, d3, ..., dn`

Além disso, existem `syscalls` utilizadas para ler e imprimir floats e doubles:

Serviço	Código ($v0)	Argumento	Retorno
Imprimir float	2	$f12	-
Imprimir double	3	$f12	-
Ler float	6	-	$f0
Ler double	7	-	$f0

As instruções para operar com ponto flutuante seguem o seguinte padrão:

`op.prec`

Onde `op` representa o mnemônico da operação e `preca` precisão ∈ {s,d}.

MIPS floating-point assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	`add.s $f2,$f4,$f6`	`$f2 = $f4 + $f6`	FP add (single precision)
	FP subtract single	`sub.s $f2,$f4,$f6`	`$f2 = $f4 - $f6`	FP sub (single precision)
	FP multiply single	`mul.s $f2,$f4,$f6`	`$f2 = $f4 × $f6`	FP multiply (single precision)
	FP divide single	`div.s $f2,$f4,$f6`	`$f2 = $f4 / $f6`	FP divide (single precision)
	FP add double	`add.d $f2,$f4,$f6`	`$f2 = $f4 + $f6`	FP add (double precision)
	FP subtract double	`sub.d $f2,$f4,$f6`	`$f2 = $f4 - $f6`	FP sub (double precision)
	FP multiply double	`mul.d $f2,$f4,$f6`	`$f2 = $f4 × $f6`	FP multiply (double precision)
	FP divide double	`div.d $f2,$f4,$f6`	`$f2 = $f4 / $f6`	FP divide (double precision)
Data transfer	load word copr. 1	`lwc1 $f1,100($s2)`	`$f1 = Memory[$s2 + 100]`	32-bit data to FP register
	store word copr. 1	`swc1 $f1,100($s2)`	`Memory[$s2 + 100] = $f1`	32-bit data to memory
Condi-tional branch	branch on FP true	`bc1t 25`	if (cond == 1) go to PC + 4 + 100	PC-relative branch if FP cond.
	branch on FP false	`bc1f 25`	if (cond == 0) go to PC + 4 + 100	PC-relative branch if not cond.
	FP compare single (eq,ne,lt,le,gt,ge)	`c.lt.s $f2,$f4`	if (`$f2 < $f4`) cond = 1; else cond = 0	FP compare less than single precision
	FP compare double (eq,ne,lt,le,gt,ge)	`c.lt.d $f2,$f4`	if (`$f2 < $f4`) cond = 1; else cond = 0	FP compare less than double precision

Acesso à memória:

- `l.s (ou l.d) $f0, 0($t2) # Carrega em $f0 o conteúdo armazenado em $t2+0`
- `s.s (ou s.d) $f0, 100($s0) # Escreve em $f0 o conteúdo armazenado em $s0+100`
- `mov.s (ou mov.d) $f0, $f2 #f0=f2`

  Obs: precisão simples ocupa 4 bytes na memória e precisão dupla, 8 bytes.

Exemplo: Suponha um vetor de ponto flutuante de precisão dupla com endereço base em `$s0` com endereço

O desvio condicional é feito em duas etapas:

- Comparação entre registradores, e o resultado é salvo em um registrador especial;
- Efetua o desvio baseado no valor do registrador especial.

Comparação e desvio

Instrução	Sintaxe	Significado
c.eq.s	c.eq.s $f2, $f4	=
c.ne.s	c.ne.s $f2, $f4	!=
c.le.s	c.le.s $f2, $f4	<=
c.lt.s	c.lt.s $f2, $f4	<
c.ge.s	c.ge.s $f2, $f4	>=
c.gt.s	c.gt.s $f2, $f4	>
c.eq.d	c.eq.d $f2, $f4	=
c.ne.d	c.ne.d $f2, $f4	!=
c.le.d	c.le.d $f2, $f4	<=
c.lt.d	c.lt.d $f2, $f4	<
c.ge.d	c.ge.d $f2, $f4	>=
c.gt.d	c.gt.d $f2, $f4	>

- Desvio
  - `bc1f label` - branch coprocessor1 label 1 if false
  - `bc1t label` - branch coprocessor1 label 1 if true

Exemplo: Faça um programa para converter a escala de temperatura Fahrenheit para Celsius.

```c
float f2c(float f){
    return (5.0/9.0)*(f-32.0);
}
```

```
    .data
    c5: .float 5.0
    c9: .float 9.0
    c32: .float 32.0
    nl: .asciiz "\n"
.text
    main:
    li $v0, 6 # cód. para ler float
    syscall # float lido está em $f0

    l.s $f1, c5 # salva o valor da constante c5 em $f1
    l.s $f2, c9 # salva o valor da constante c9 em $f2
    l.s $f3, c32 # salva o valor da constante c32 em $f3

    sub.s $f12, $f0, $f3 # f12 = F - 32
    mul.s $f12, $f12, $f1 # f12 = (F-32) * 5
    div.s $f12, $f12, $f2 # f12 = (F-32) * 5/9

    li $v0, 2 # código para imprimir float (em $f12) na tela
    syscall

    li $v0, 4 # código para imprimir string (em $a0) na tela
    la $a0, nl  # carrega o valor de nl em a0
    syscall

    li $v0, 10 # código para encerrar o programa
    syscall
```

Instruções MIPS

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow detected
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	addu $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow undetected
	add immediate unsigned	addiu $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	mfc0 $s1,$epc	$s1 = $epc	Copy Exception PC + special regs
	multiply	mult $s2,$s3	Hi, Lo = $s2 × $s3	64-bit signed product in Hi, Lo
	multiply unsigned	multu $s2,$s3	Hi, Lo = $s2 × $s3	64-bit unsigned product in Hi, Lo
	divide	div $s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Lo = quotient, Hi = remainder
	divide unsigned	divu $s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Unsigned quotient and remainder
	move from Hi	mfhi $s1	$s1 = Hi	Used to get copy of Hi
	move from Lo	mflo $s1	$s1 = Lo	Used to get copy of Lo
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store conditional word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half atomic swap
	load upper immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	AND	AND $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	OR	OR $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	NOR	NOR $s1,$s2,$s3	$s1 = ~ ($s2 |$s3)	Three reg. operands; bit-by-bit NOR
	AND immediate	ANDi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND with constant
	OR immediate	ORi $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1=0	Compare < constant; two's complement
	set less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1=0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call