Universidade Federal
do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Arquitetura de Computadores I – Turmas 01 e 02 (EARTE) – 2021/2
Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br

Laboratório V – Aritmética de Ponto Flutuante

1. Objetivo

- Conhecer as operações de ponto flutuante e sua representação numérica;
- Usar as instruções de ponto flutuante.

2. Introdução

O padrão IEEE-754 reserva vários padrões de bits para ter um significado especial. Em outras palavras, nem todos os padrões de bits representam algum número, conforme quadro a seguir.

Special bit patterns in IEEE-754

Sign bit	Exponent	Significand	Comment
x	0..0	0..0	Zero
x	0..0	not all zeros	Denormalized number
0	1..1	0..0	Plus infinity (+inf)
1	1..1	0..0	Minus infinity (-inf)
x	1..1	not all zeros	Not a Number (NaN)

Infinito significa algo grande demais para ser representado. Um estouro de representação pode retornar um + inf ou um -inf. Algumas operações no infinito retornam outro infinito como resultado. Um resumo dessa representação encontra-se na Tabela 1.

Table 1: Some operations with infinity

Operation	Result	Comment
x + (inf)	inf	x finite
x - (+inf)	-inf	x finite
(+inf) + (+inf)	+inf	
(-inf) + (-inf)	-inf	
x * (+inf)	+inf if x>0, -inf otherwise	x nonzero

Pode haver um zero positivo se o bit de sinal for 0 e um zero negativo (bit de sinal é 1). Números desnormalizados estão incluídos no padrão IEEE-754 para lidar com casos de estouro negativo de expoente (números muito pequenos). Um NaN (às vezes denotado por nan) é usado para representar um resultado indeterminado. Existem dois tipos de NaNs: sinalização e silêncio. O padrão de bits no significando é usado para diferenciar entre eles, e é dependente da implementação. Um NaN de sinalização pode ser usado, por exemplo, para variáveis não inicializadas. Observe que qualquer operação em um NaN de sinalização terá como resultado, um NaN silencioso. Operar em um NaN silencioso simplesmente retorna outro NaN sem gerar nenhuma exceção, vide Tabela 2.

Table 2: Some operations which produce a quiet NaN

Operation	Comment
`x + (NaN)`	Any operation on a quiet NaN (addition in this example)
`(+inf) + (-inf)`	
`0 * ( inf)`	
`0/0`	
`inf/inf`	
`x%0`	The remainder of division by 0
$\sqrt{x}$, `x < 0`	

Atividade 1
Usando o simulador MARS:

- Declare as variáveis Zero.s, PlusInf.s, MinusInf.s, PlusNaN.s, MinusNaN, inicializadas com os padrões de bits correspondendo a zero, mais infinito, menos infinito, NaN positivo, NaN negativo em representação de precisão simples.
- Declare as variáveis Zero.d, PlusInf.d, MinusInf.d, NaN.d, inicializadas com os padrões de bits correspondendo a zero, mais infinito, menos infinito, NaN positivo, NaN negativo em representação de precisão dupla.
- Carregue essas variáveis em registradores de ponto flutuante, começando com $f0
- Imprima, começando com $f0, o conteúdo dos registradores onde as variáveis foram carregadas; imprime um caractere de nova linha (\n) após cada valor.

Execute o programa e preencha a tabela a seguir com o nome de cada variável e o respectivo valor impresso após a execução:

Variável	Saída impressa

Qual é o padrão de bits para o maior número possível de ponto flutuante de precisão única? Escreva em hexadecimal.

Atividade 2

Nesta atividade você deverá criar um programa que calcula o fatorial de um número inteiro, representado usando notação de números de ponto flutuante, conforme passos a seguir:

- Solicite ao usuário que insira um número inteiro;
- Verifique se o número inserido é negativo: se for negativo, imprima uma mensagem de erro e solicite o usuário para entrar novamente com um número inteiro positivo
- Realizar a operação ‘FactorialSingle’, cujo parâmetro será o número lido do usuário convertido em ponto flutuante de precisão simples. A operação deve retornar o fatorial (em PF precisão simples) desse número
- Imprimir o valor retornado por ‘FactorialSingle’

Execute o 'FactorialSingle' para completar o plano de testes a seguir. Use notação científica normalizada para representar a saída impressa por 'FactorialSingle'.

Número	Fatorial (Inteiro)	Fatorial PF (single)
0		
5		
10		
15		
20		
40		

A seguir, responda:
a) Por que alguns dos resultados impressos são negativos?
b) Quais são os valores máximos da entrada para os quais a saída correta ainda é impressa?
c) Usando algum outro método ou linguagem de programação, calcule o valor exato de 20 e compare com o valor impresso pelo seu programa no MARS. Quais são esses valores? Por quê os valores são diferentes?

Atividade 3
Na Atividade 2 você usou a conversão de inteiro para ponto flutuante (cvt.s.w). Vamos agora tentar uma conversão de PF para inteiro, conforme instruções a seguir:

- Após de imprimir o valor retornado por 'FactorialSingle', converta esse valor em um número inteiro e imprima-o também.

Execute esse novo programa e conclua o próximo plano de teste. Anote o valor impresso para o fatorial como está, não use notação científica neste momento.

Número	Fatorial (Impressão como Inteiro)	Fatorial (Impressão como PF simples)
0		
5		
10		
11		
12		
13		
14		
15		

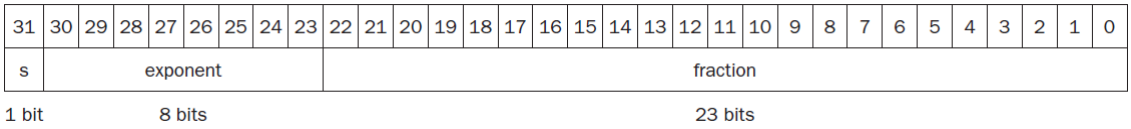
Destaque os casos em que a saída de inteiro é um número diferente da saída de ponto flutuante. **A** operação de conversão produz um inteiro com ou sem sinal? Explique

4. Execução
- Grupos de até 3 (três) alunos;
- Submissão até 17/12 (23:59h);

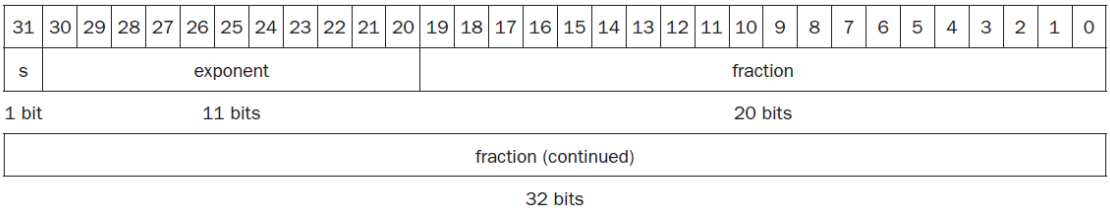
5. Apêndice – PF no MARS

A aritmética de Ponto Flutuante é implementada pelo coprocessador 1 (*Coproc 1* no MARS) da arquitetura MIPS. O coprocessador possui 32 registradores de 32 bits, numerados de 0 a 31 ($f0 a $f31). Os valores armazenados nestes registradores seguem o padrão IEEE-754 (ver material do curso).

Para permitir o armazenamento de valores em precisão dupla (64 bits), a arquitetura usa o artifício de agrupar pares de registradores subsequentes, isto é, $f0 com $f1, $f2 com $f3, e assim, sucessivamente. Assim, a arquitetura oferece "virtualmente" 16 registradores de 64 bits, nos quais o valor armazenado terá sempre seus 32 bits de mais alta ordem armazenados num registrador par e os 32 de mais baixa ordem num registrador ímpar. A representação em IEEE-754 para 32 e 64 bits é mostrada abaixo:



Representação IEEE-754 em precisão simples (32 bits)



Representação IEEE-754 em precisão dupla (64 bits)

Para simplificar as coisas, operações em ponto flutuante usam sempre registradores numerados e podem ser executadas em precisão simples (32 bits) ou precisão dupla (64 bits). A diferença das instruções MIPS é determinada por um sufixo após o mnemônico da instrução, por exemplo, as instruções add.s e add.d representam adições em precisão simples (.s) e precisão dupla (.d), respectivamente.

O conjunto de instruções MIPS oferece, além das instruções aritméticas, comparações, desvios, movimentação de dados da (*load*) e para (*store*) a memória, conversões de formatos de ponto-flutuante (32 para 64 bits e vice-versa) e conversão de inteiro para ponto flutuante e vice-versa. Enquanto nas comparações envolvendo inteiros um dos registradores de propósito geral pode ser definido como destino da execução, no caso de ponto flutuante, uma comparação irá implicitamente determinar ("setar") uma *flag*. Esta *flag* poderá então ser usada para testar se um desvio é realizado ou não numa instrução de desvio condicional.

Instruction	Comment
`mfc1 Rdest, FPsrc`	Move the content of floating-point register `FPsrc` to `Rdest`
`mtc1 Rsrc, FPdest`	Integer register `Rsrc` is moved to floating-point register `FPdest`
`mov.x FPdest, FPsrc`	Move floating-point register `FPsrc` to `FPdest`
`lwc1 FPdest, address`	Load word from `address` in register `FPdest`[a]
`swc1 FPsrc, address`	Store the content of register `FPsrc` at `address`[b]
`add.x FPdest, FPsrc1, FPsrc2`	Add single precision

Instruction	Comment
`sub.x FPdest, FPsrc1, FPsrc2`	Subtract `FPsrc2` from `FPsrc1`
`mul.x FPdest, FPsrc1, FPsrc2`	Multiply
`div.x FPdest, FPsrc1, FPsrc2`	Divide `FPsrc1` by `FPsrc2`
`abs.s FPdest, FPsrc`	Store the absolute value of `FPsrc` in `FPdest`
`neg.x FPdest, FPsrc`	Negate number in `FPsrc` and store result in `FPdest`
`c.eq.x FPsrc1, FPscr2`	Set the floating-point condition flag to true if the two registers are equal
`c.le.x FPsrc1, FPsrc2`	Set the floating-point condition flag to true if `FPsrc1` is less than or equal to `FPsrc2`
`c.lt.x FPsrc1, FPsrc2`	Set the floating-point condition flag to true if `FPsrc1` is less than `FPsrc2`
`bc1t label`	Branch if the floating-point condition flag is true
`bc1f label`	Branch if the floating-point condition flag is false
`cvt.x.w FPdest, FPsrc`	Convert the integer in `FPsrc` to floating-point
`cvt.w.x FPdest, FPsrc`	Convert the floating-point number in `FPsrc` to integer
`cvt.d.s FPdest, FPsrc`	Convert the single precision number in `FPsrc` to double precision and put the result in `FPdest`
`cvt.s.d FPdest, FPsrc`	Convert the double precision number in `FPsrc` to single precision and put the result in `FPdest`