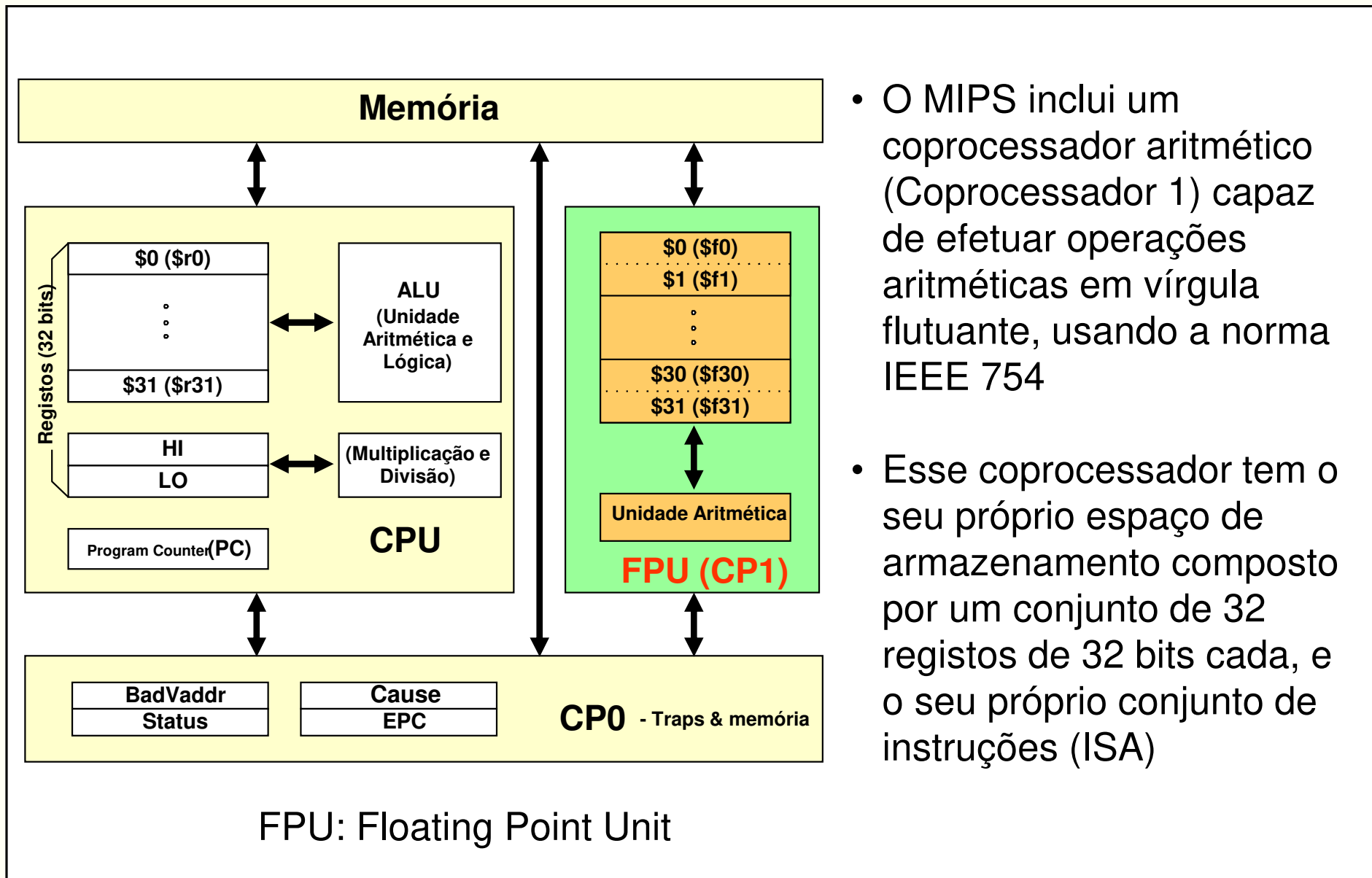


Aula 8

- Unidade de vírgula flutuante do MIPS (coprocessador 1)
 - Instruções da FPU do MIPS
 - Análise de um exemplo de tradução de C para assembly

Bernardo Cunha, José Luís Azevedo

Cálculo em Vírgula Flutuante no MIPS



- O MIPS inclui um coprocessador aritmético (Coprocessador 1) capaz de efetuar operações aritméticas em vírgula flutuante, usando a norma IEEE 754
- Esse coprocessador tem o seu próprio espaço de armazenamento composto por um conjunto de 32 registros de 32 bits cada, e o seu próprio conjunto de instruções (ISA)

Vírgula Flutuante no MIPS – registos

- Os registos do coprocessador 1 são designados por **\$fn**, em que o índice **n** toma valores entre 0 e 31 (\$f0, \$f1, \$f2, ...)
- Cada par de registos consecutivos [**\$fn,\$fn+1**] (**com n par**) pode funcionar como um registo de 64 bits para armazenar valores em **precisão dupla**.
- A referência ao conjunto de 2 registos faz-se sempre indicando como operando o **registo par** (\$f0, \$f2, \$f4,...)
- **Apenas os registos de índice par** podem ser usados no contexto das instruções

Vírgula Flutuante no MIPS – instruções aritméticas

<code>abs.p</code>	<code>FPdst, FPsrc</code>	<code>#Absolute Value</code>
<code>neg.p</code>	<code>FPdst, FPsrc</code>	<code>#Negate</code>
<code>div.p</code>	<code>FPdst, FPsrc1, FPsrc2</code>	<code>#Divide</code>
<code>mul.p</code>	<code>FPdst, FPsrc1, FPsrc2</code>	<code>#Multiply</code>
<code>add.p</code>	<code>FPdst, FPsrc1, FPsrc2</code>	<code>#Addition</code>
<code>sub.p</code>	<code>FPdst, FPsrc1, FPsrc2</code>	<code>#Subtract</code>

- O sufixo `.p` representa a **precisão** com que é efetuada a operação (simples ou dupla); na instrução é substituído pelas letras `.s` ou `.d` respetivamente
- Exemplos:

`add.s $f0, $f4, $f6` `#$f0=$f4 + $f6`

`div.d $f4, $f0, $f8` `#$f4 ($f5)=$f0 ($f1) / $f8 ($f9)`

Vírgula Flutuante no MIPS – conversão entre tipos

<code>cvt.d.s</code>	<code>FPdst,FPsrc</code>	<code>#Convert Float to Double</code>
<code>cvt.d.w</code>	<code>FPdst,FPsrc</code>	<code>#Convert Integer to Double</code>
<code>cvt.s.d</code>	<code>FPdst,FPsrc</code>	<code>#Convert Double to Float</code>
<code>cvt.s.w</code>	<code>FPdst,FPsrc</code>	<code>#Convert Integer to Float</code>
<code>cvt.w.d</code>	<code>FPdst,FPsrc</code>	<code>#Convert Double to Integer</code>
<code>cvt.w.s</code>	<code>FPdst,FPsrc</code>	<code>#Convert Float to Integer</code>

- A letra mais à direita especifica o formato original; a letra do meio, especifica o formato do resultado - **s**: float (single), **d**: double, **w**: inteiro
- As **conversões** entre tipos de representação são efetuadas pela FPU: **os registos operando e destino das instruções são obrigatoriamente registos da FPU**

Conversão entre tipos – exemplos

\$f0=0xC0D00000 = 11000000110100000000000000000000
= 11000000110100000000000000000000
= -1.625 x 2² = -6.5

```
cvt.d.s $f6,$f0 #Convert Float to Double
```

$$\mathbf{E} = (129-127) + 1023 = 1025 = 10000000001_2$$

\$f6=0x00000000 \$f7=**1** 100000000001 1010000...0

\$f6=0x00000000 \$f7=0xC01A0000

```
cvt.w.s $f8,$f0 #Convert Float to Integer
```

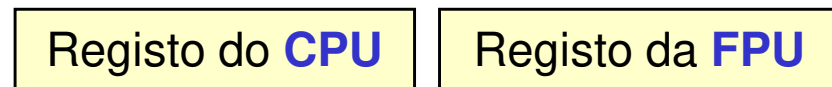
$$\mathbf{Exp} = (129-127) = 2$$
$$\mathbf{Val} = -1.625 \times 2^2 = -6.5$$

Resultado: (int) (-6.5) = trunc (-6.5) = -6

\$f8=0xFFFFFFFFFA (-6 em complemento para 2)

Vírgula Flutuante no MIPS – instruções de transferência

- **Transferência de informação** entre registros do CPU e da FPU, e entre registros da FPU



mtc1	CPUsrc, FPdst	#Move <u>to</u> Coprocessor 1 #Ex: mtc1 \$t0, \$f4
mfc1	CPUdst, FPsrc	#Move <u>from</u> Coprocessor 1 #Ex: mfc1 \$a0, \$f6
mov.s	FPdst, FPsrc	#Move from FPsrc to FPdst (single) #Ex: mov.s \$f4, \$f8
mov.d	FPdst, FPsrc	#Move from FPsrc to FPdst (double) #Ex: mov.d \$f2, \$f0

- Estas instruções copiam o conteúdo integral do registro fonte para o registro destino
- **Não fazem qualquer tipo de conversão entre tipos de informação**

Vírgula Flutuante no MIPS – instruções de transferência

- **Transferência de informação** entre registos da FPU e a memória

Registo da FPU	Endereço de memória	
l.s	FPdst , offset (CPUreg)	#Load Float from memory #Ex: l.s \$f0,4(\$a0)
s.s	FPsrc , offset (CPUreg)	#Store Float into memory #Ex: s.s \$f0,0(\$a0)
l.d	FPdst , offset (CPUreg)	#Load Double from memory #Ex: l.d \$f4,8(\$a1)
s.d	FPsrc , offset (CPUreg)	#Store Double into memory #Ex: s.d \$f4,16(\$t0)

Instruções nativas (só muda a mnemónica):

lwc1	FPdst , offset (CPUreg)	#Load Float from memory
swc1	FPsrc , offset (CPUreg)	#Store Float into memory
ldc1	FPdst , offset (CPUreg)	#Load Double from memory
sdc1	FPsrc , offset (CPUreg)	#Store Double into memory

Vírgula Flutuante no MIPS – Manipulação de constantes

- Nas instruções da FPU do MIPS os operandos têm que residir em registros internos, o que significa que **não há suporte para a manipulação direta de constantes**. Como lidar então com operandos que são constantes?
- **Método 1:**
 - Determinar, manualmente, o valor que codifica a constante (32 bits para precisão simples ou 64 bits para precisão dupla)
 - Carregar essa constante em 1 ou 2 registros do CPU e copiar o(s) seu(s) valor(es) para o(s) registro(s) da FPU
- **Método 2:**
 - Usar as directivas “**.float**” ou “**.double**” para definir em memória o valor da constante: 32 bits (**.float**) ou 64 bits (**.double**)
 - Ler o valor da constante da memória para um registro da FPU usando as instruções de acesso à memória (**l.s** ou **l.d**)

Vírgula Flutuante no MIPS – instruções de decisão

- A tomada de decisões envolvendo quantidades em vírgula flutuante realiza-se de forma distinta da utilizada para o mesmo tipo de operação envolvendo quantidades inteiras
- Para quantidades em vírgula flutuante são necessárias duas instruções em sequência: uma **comparação das duas quantidades, seguida da decisão** (que usa a informação produzida pela comparação):
 - A instrução de comparação coloca a **True** ou **False** uma *flag* (1 bit), dependendo de a condição em comparação ser verdadeira ou falsa, respetivamente
 - Em **função do estado dessa flag** a instrução de decisão (instrução de salto) pode alterar a sequência de execução

Cálculo em Vírgula Flutuante no MIPS

- Instruções de comparação:

`c.xx.s FPUreg1, FPUreg2` # compare float

`c.xx.d FPUreg1, FPUreg2` # compare double

Em que `xx` pode ser uma das seguintes condições:

`EQ` - equal

`LT` - less than

`LE` - less or equal

Exemplos:

`c.eq.s $f0, $f2` / `c.le.d $f4, $f8`

- Instruções de salto:

`bc1t label` # branch if true

`bc1f label` # branch if false

Vírgula Flutuante no MIPS – instruções de decisão

```
float a, b;  
...  
  
if( a > b)  
    a = a + b;  
else  
    a = a - b;
```

```
# a: $f0  
# b: $f2  
...  
if:    c.le.s $f0, $f2          # if(a > b)  
      bc1t   else             # {  
      add.s  $f0, $f0, $f2     #     a = a + b;  
      j      endif            # }  
else:  sub.s  $f0, $f0, $f2     # else  
      #      a = a - b;  
endif:...
```

Convenções de utilização dos registos

- Registos para **passar parâmetros** para sub-rotinas (do tipo float ou double):
 - **\$f12** (\$f13), **\$f14** (\$f15), por esta ordem
- Registos para **devolução de resultados** das sub-rotinas:
 - **\$f0** (\$f1)
- Registos que **podem** ser livremente usados e alterados pelas sub-rotinas ("caller-saved"):
 - **\$f0** (\$f1) a **\$f18** (\$f19)
- Registos que **não podem** ser alterados pelas sub-rotinas ("callee-saved"):
 - **\$f20** (\$f21) a **\$f30** (\$f31)

Tradução C / Assembly – Exemplo 1

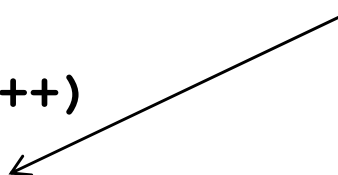
```
#define SIZE 25
double average(double *, int);

void main(void)
{
    double array[SIZE];
    double avg;
    ...
    avg = average( array, SIZE );
    print_double( avg );    // syscall 3
}
```

```
double average(double *v, int N)
{
    double sum = 0.0;
    int i;

    for(i = 0; i < N; i++)
        sum += v[i];
    return sum / (double)N;
}
```

Conversão entre tipos
(inteiro para double)



Tradução C / Assembly – Exemplo 1

```
void main(void)
{
    static double array[SIZE];
    double avg;
    ...
    avg = average( array, SIZE );
    print_double( avg );    // syscall 3
}
```

```
double average(double *, int)
```

```
        .data
array:   .space 200                # 8*SIZE (alinhado múltiplo 8)
        .eqv SIZE,25
        .text
        .globl main               # avg: $f12
main:    ...                      # Salva guarda $ra
        la      $a0, array        #
        li      $a1, SIZE         #
        jal     average           #
        mov.d   $f12, $f0         # avg = average(array, SIZE)
        li      $v0, 3            #
        syscall                    # print_double(avg)
        ...                      # Repõe $ra
        jr      $ra              #
```

Tradução C / Assembly – Exemplo 1

```
double average(double *v, int N)
{
    double sum = 0.0;
    int i;
    for(i = 0; i < N; i++)
        sum += v[i];
    return sum / (double)N;
}
```

```
# sum: $f0 / tmp1: $f4 / i: $t0 / tmp2: $t1
average: mtc1      $0,    $f0          #
          cvt.d.w  $f0,    $f0          # sum = 0.0
          li       $t0,    0            # i = 0
for:      bge      $t0,    $a1, endf     # while(i < N) {
          sll      $t1,    $t0, 3        # tmp = i * 8
          addu     $t1,    $t1, $a0      # $t1 = &v[i]
          l.d      $f4,    0($t1)        # $f4 = v[i]
          add.d    $f0,    $f0, $f4      # sum += v[i]
          addi     $t0,    $t0, 1        # i++
          j        for                  # }
endf:     mtc1      $a1,    $f4          #
          cvt.d.w  $f4,    $f4          # $f4 = (double)N
          div.d    $f0,    $f0, $f4      # return sum / (double)N
          jr       $ra                  #
```


Tradução C / Assembly – Exemplo 2

```
float fun(float, int);

void main(void)
{
    float res;

    res = fun( 12.5E-2, 2 );
    print_float( res );    // syscall 2
}
```

```
float fun(float a, int m)
{
    float val;
    if( a >= -5.6 )
        val = (float)m * (a - 32.0);
    else
        val = 0.0;
    return val;
}
```

Tradução C / Assembly – Exemplo 2

```
void main(void)
```

```
{
```

```
    float  res;
```

```
    res = fun( 12.5E-2, 2 );
```

```
    print_float( res );    // syscall 2
```

```
}
```

```
float fun(float a, int k)
```

```
.data
```

```
k1:    .float 12.5E-2      # 12.5 x 10-2
```

```
k2:    .float -5.6
```

```
k3:    .float 32.0
```

```
k4:    .float 0.0
```

```
.text
```

```
.globl main                # res: $f12
```

```
main:    ...                # salvaguarda $ra
```

```
    la    $t0, k1
```

```
    l.s   $f12, 0($t0)      # $f12 = 12.5E-2
```

```
    li    $a0, 2            # $a0 = 2
```

```
    jal   fun               #
```

```
    mov.s $f12, $f0         # res = fun(12.5E-2, 2)
```

```
    li    $v0, 2            #
```

```
    syscall                # print_float(res)
```

```
    ...                    # repõe $ra
```

```
    jr    $ra              #
```

Tradução C / Assembly – Exemplo 2

```
float fun(float a, int m)
{
    float val;
    if( a >= -5.6)
        val = (float)m * (a - 32.0);
    else
        val = 0.0;
    return val;
}
```

```
.data
k1: .float 12.5E-2
k2: .float -5.6
k3: .float 32.0
k4: .float 0.0
```

```
# val: $f2 / a: $f12 / m: $a0
```

```
fun:    la        $t0, k2
        l.s       $f0, 0($t0)           # $f0 = -5.6
        c.lt.s    $f12,$f0              # if( a >= -5.6 )
        bclt      else                  # {
        la        $t0, k3
        l.s       $f2, 0($t0)           # val = 32.0
        sub.s     $f2, $f12, $f2        # val = a - 32.0
        mtc1      $a0, $f0              # $f0 = m
        cvt.s.w   $f0, $f0              # $f0 = (float)m
        mul.s     $f2, $f0, $f2         # val = (float)m * val
        j         endif                 # }
else:   la        $t0, k4                # else
        l.s       $f2, 0($t0)           # val = 0.0
endif:  mov.s     $f0, $f2               # return val;
        jr        $ra                   #
```

Exercícios

- Considere que o conteúdo dos dois seguintes registros da FPU representam a codificação de duas quantidades reais no formato IEEE754 precisão simples:

- `$f0 = 0x416A0000`
- `$f2 = 0xC0C00000`

Calcule o resultado das instruções seguintes, apresentando o resultado em hexadecimal:

- `abs.s $f4, $f2` # `$f4 = abs($f2)`
- `neg.s $f6, $f0` # `$f6 = neg($f0)`
- `sub.s $f8, $f0, $f2` # `$f8 = $f0 - $f2`
- `sub.s $f10, $f2, $f0` # `$f10 = $f2 - $f0`
- `add.s $f12, $f0, $f2` # `$f12 = $f0 + $f2`
- `mul.s $f14, $f0, $f2` # `$f14 = $f0 * $f2`
- `div.s $f16, $f0, $f2` # `$f16 = $f0 / $f2`
- `div.s $f18, $f2, $f0` # `$f18 = $f2 / $f0`
- `cvt.d.s $f20, $f2` # Convert single to double
- `cvt.w.s $f22, $f0` # Convert single to integer