Mestrado em Informática

2009/10

A.J.Proença

Tema

Os Números nos Sistemas de Computação


```
double recip(int denom)
{
  return 1.0/(double) denom;
}

void do_nothing() {}

void test1(int denom)
{
  double r1, r2;
  int t1;
  r1 = recip(denom);
  r2 = recip(denom*2-10);
  t1 = r1 == r2;
  do_nothing();
  printf("test1 t1: r1 %f %c= r2 %f\n", r1, t1 ? '=' : '!', r2);
}

void test2(int denom)
{
  double r1, r2;
  int t2;
  r1 = recip(denom);
  r2 = recip(denom*2-10);
  do_nothing();
  t2 = r1 == r2;
  printf("test2 t2: r1 %f %c= r2 %f\n", r1, t2 ? '=' : '!', r2);
}
```

```
int main()
{
  int denom=10;

  test1(denom);
  test2(denom);
}
```

Será 1/10 sempre igual a 1/10?


```
double recip(int denom)
{
  return 1.0/(double) denom;
}

void do_nothing() {}

void test1(int denom)
{
  double r1, r2;
  int t1;
  r1 = recip(denom);
  r2 = recip(denom*2-10);
  t1 = r1 == r2;
  do_nothing();
  printf("test1 t1: r1 %f %c= r2 %f\n", r1, t1 ? '=' : '!', r2);
}

void test2(int denom)
{
  double r1, r2;
  int t2;
  r1 = recip(denom);
  r2 = recip(denom*2-10);
  do_nothing();
  t2 = r1 == r2;
  printf("test2 t2: r1 %f %c= r2 %f\n", r1, t2 ? '=' : '!', r2);
}
```

```
int main()
{
  int denom=10;

  test1(denom);
  test2(denom);
}
```

```
test1 t1: r1 0.100000 != r2 0.100000
test2 t2: r1 0.100000 == r2 0.100000
```

Porquê?

- Erro no algoritmo?
- Erro na codificação?
- Erro na compilação? ... provavelmente...

  então:
  - analisar o código gerado
  - e procurar as diferenças que causaram a anomalia
    - comparar código desmontado de `test1` e `test2`

```
void test1(int denom)
{ ...
  ...
  t1 = r1 == r2;
  do_nothing();
  printf(...);
}
```

```
void test2(int denom)
{ ...
  ...
  do_nothing();
  t2 = r1 == r2;
  printf(...);
}
```

Olhar de novo para o código...

```
double recip(int denom)
{
  return 1.0/(double) denom;
}

void do_nothing() {}

void test1(int denom)
{
  double r1, r2;
  int t1;
  r1 = recip(denom);
  r2 = recip(denom*2-10);
  t1 = r1 == r2;
  do_nothing();
  printf("test1 t1: r1 %f %c= r2 %f\n", r1, t1 ? '=' : '!', r2);
}

void test2(int denom)
{
  double r1, r2;
  int t2;
  r1 = recip(denom);
  r2 = recip(denom*2-10);
  do_nothing();
  t2 = r1 == r2;
  printf("test2 t2: r1 %f %c= r2 %f\n", r1, t2 ? '=' : '!', r2);
}
```

```
int main()
{
  int denom=10;

  test1(denom);
  test2(denom);
}
```


Análise do código desmontado de `test1` (1)

```
00401090 <_test1>:
401090:  55                  push   %ebp
401091:  89 e5               mov    %esp,%ebp
401093:  83 ec 28            sub    $0x28,%esp
401096:  89 5d fc            mov    %ebx,0xfffffffc(%ebp)
401099:  8b 5d 08            mov    0x8(%ebp),%ebx
40109c:  89 1c 24            mov    %ebx,(%esp,1)
40109f:  8d 5c 1b f6         lea    0xfffffff6(%ebx,%ebx,1),%ebx
4010a3:  e8 b8 ff ff ff      call   401060 <_recip>
4010a8:  89 1c 24            mov    %ebx,(%esp,1)
4010ab:  dd 5c 24 04         fstpl  0x4(%esp,1)
4010af:  e8 ac ff ff ff      call   401060 <_recip>
4010b4:  dd 44 24 04         fldl   0x4(%esp,1)
4010b8:  d9 c9               fxch   %st(1)
4010ba:  dd 54 24 10         fstl   0x10(%esp,1)
4010be:  da e9               fucompp
4010c0:  df e0               fnstsw %ax
4010c2:  9e                  sahf
4010c3:  0f 9b c2            setnp  %dl
4010c6:  0f 94 c0            sete   %al
4010c9:  20 d0               and    %dl,%al
4010cb:  0f b6 d8            movzbl %al,%ebx
4010ce:  e8 9d ff ff ff      call   401070 <_do_nothing>
4010d3:  c7 04 24 75 10 40 00        movl   $0x401075,(%esp,1)
4010da:  31 c0               xor    %eax,%eax
4010dc:  85 db               test   %ebx,%ebx
4010de:  0f 95 c0            setne  %al
4010e1:  48                  dec    %eax
4010e2:  83 e0 e4            and    $0xffffffe4,%eax
4010e5:  83 c0 3d            add    $0x3d,%eax
4010e8:  89 44 24 0c         mov    %eax,0xc(%esp,1)
4010ec:  e8 ff 04 00 00      call   4015f0 <_printf>
4010f1:  8b 5d fc            mov    0xfffffffc(%ebp),%ebx
4010f4:  89 ec               mov    %ebp,%esp
4010f6:  5d                  pop    %ebp
4010f7:  c3                  ret
4010f8:  74 65               je     40115f <_test2+0x3f>
4010fa:  73 74               jae    401170 <_test2+0x50>
4010fc:  32 20               xor    (%eax),%ah
4010fe:  74 32               je     401132 <_test2+0x12>
401100:  3a 20               cmp    (%eax),%ah
401102:  72 31               jb     401135 <_test2+0x15>
401104:  20 25 66 20 25 63           and    %ah,0x63252066
40110a:  3d 20 72 32 20      cmp    $0x20327220,%eax
40110f:  25 66 0a 00 8d      and    $0x8d000a66,%eax
401114:  b6 00               mov    $0x0,%dh
401116:  00 00               add    %al,(%eax)
401118:  00 8d bc 27 00 00           add    %cl,0x27bc(%ebp)
```


Análise do código desmontado de `test1` (1)

```
push   %ebp
mov    %esp,%ebp
sub    $0x28,%esp
mov    %ebx,0xfffffffc(%ebp)
```

bloco de arranque da função

```
mov    0x8(%ebp),%ebx
```

valor do arg recebido <denom> em %ebx

```
mov    %ebx,(%esp,1)
```

topo da stack <- valor do arg <denom> para <recip>

```
lea    0xfffffff6(%ebx,%ebx,1),%ebx
```

o próximo arg (= denom + denom - 10)

```
call   401060 <_recip>
```

calcula o recíproco de 10

```
mov    %ebx,(%esp,1)
```

topo da stack <- o novo valor do arg para <recip>

```
fstpl  0x4(%esp,1)
```

r1=val_ret: pop/sto %st(0) na mem (reduz a 64b); 2º arg p/ <printf>

```
call   401060 <_recip>
```

calcula o novo recíproco (que é 10 outra vez)

```
fldl   0x4(%esp,1)
```

recupera r1, i.e., push para %st(0) - valor de 64b -> 80b

```
fxch   %st(1)
```

troca %st0 (r1) com %st1 (o novo recíproco)

```
fstl   0x10(%esp,1)
```

r2=val_ret: store %st(0) na mem (reduz a 64b); 4º arg p/ <printf>

```
fucompp
```

compara+pop os 2 reg, com r1(64b->80b) e r2(80b)

Análise do código desmontado de `test1` (2)

```
mov    0x8(%ebp),%ebx
mov    %ebx,(%esp,1)
lea    0xfffffff6(%ebx,%ebx,1),%ebx
call   401060 <_recip>
mov    %ebx,(%esp,1)
fstpl  0x4(%esp,1)
call   401060 <_recip>
fldl   0x4(%esp,1)
fxch   %st(1)
fstl   0x10(%esp,1)
fucompp
```

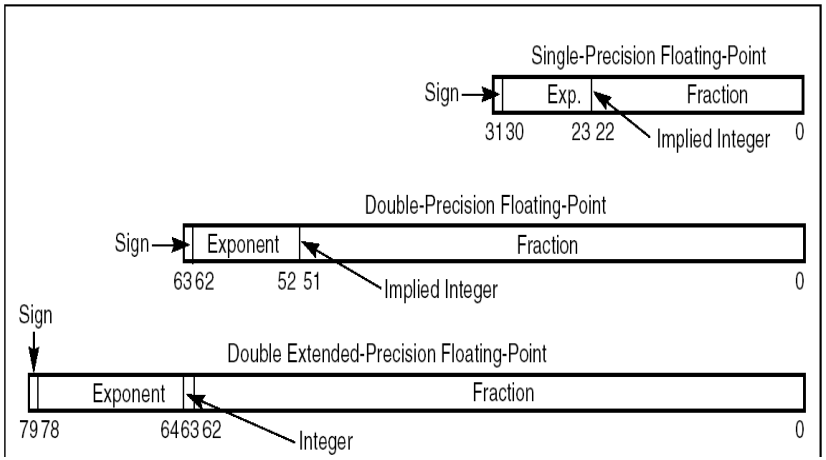
Análise do código desmontado de `test2`

```
call   401060 <_recip>        ; calcula o recíproco de 10
mov    %ebx,(%esp,1)
fstpl  0x4(%esp,1)            ; guarda na mem o valor de ret da func
                                (r1 reduzido a 64b)
call   401060 <_recip>        ; calcula o novo recíproco
                                (que neste caso é 10 outra vez)
fstpl  0x10(%esp,1)           ; guarda tb na mem o valor retorno da função
                                (r2 reduzido a 64b)
call   401070 <_do_nothing>
fldl   0x4(%esp,1)            ; recupera r1
fldl   0x10(%esp,1)           ; recupera r2
movl   $0x4010f8,(%esp,1)
fucompp                       ; compara r1:r2
                                (ambos ex-80b arredondados para 64b)
```


Análise do código: conclusões

- detectada 1 diferença fundamental:
  - a comparação em **`test1`** é feita entre
    - **`r1`**, calcul c/ 80b, armazen c/ 64b, recup c/ ext a 80b
    - **`r2`**, valor calculado e mantido em 80b
  - e em test2: r1 e r2 são ambos 80b->64b->80b
  - quando ocorre esta salvag autom de registos?
    - nesta versão do compilador, os valores em reg fp são salvaguardados antes de invocar a função
- a diferença é crítica?
  - será se a representação binária do valor (IEEE 754) necessitar mais que os 64b da precisão dupla
  - o valor decimal "0.1" necessita?...

Representação de reais no IA32



Análise da representação binária do valor decimal "0.1"

- $0.1_{10} = 0.0(0011)_2$ **(valor normalizado)**
  - prec dupla estendida, 80b (**`frac`** com 63b + 1b):
    - $0.0(0011)_2 * 2^0 =$

$1.1001100110011001100110011001100110011001100110011001100110011001101 * 2^{-4}$

  - prec dupla, de 80b arredond p/ 64b (frac com 52b + 1b):
    - $0.0(0011)_2 * 2^0 =$

$1.1001100110011001100110011001100110011001100110011001100110011010 * 2^{-4}$

  - de prec dupla 64b estend p/ 80b (**`frac`** com 63b + 1b) :
    - $0.0(0011)_2 * 2^0 =$

$1.1001100110011001100110011001100110011001100110011001100110000000000000 * 2^{-4}$