Instruções de Ponto Flutuante

Noemi Rodriguez
Ana Lúcia de Moura

http://www.inf.puc-rio.br/~inf1018

Ponto Flutuante em x86-64

Arquiteturas anteriores a x86-64

- conjunto de instruções ("x87") herdadas da implementação da UPF como um *coprocessador*
- pilha de 8 registradores de 80 bits (formato não padrão IEEE)
- dificuldade para geração de código

Arquitetura x86-64

- Código de ponto flutuante baseado no conjunto de registradores e operações **SSE** (*Streaming SIMD Extensions*)

Extensão SSE

Implementada originalmente para prover suporte a processamento gráfico e de imagens

SSE3: 16 registradores (**%xmm0** a **%xmm15**) de 128 bits

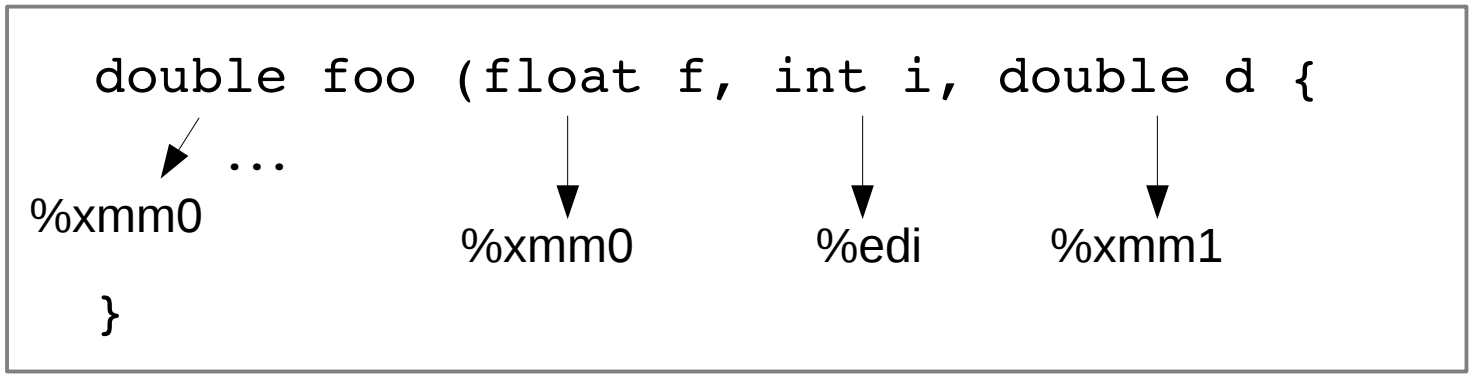
- **formato *packed***: registrador armazena um **vetor** de elementos
- **formato escalar**: um único valor de ponto flutuante (float ou double) na parte baixa do registrador

Chamadas de Procedimentos

Até 8 argumentos de ponto flutuante passados em registradores (**%xmm0** a **%xmm7**)

Todos os registradores %xmm podem ser **sobrescritos** pela função chamada!

Valor de retorno de ponto flutuante em **%xmm0**



Movimentação de Dados


```
double foo(double a, double b){
  return b;
}
```

```
foo:
  movsd %xmm1, %xmm0
  ret
```

```
float boo (float *a) {
  return *a;
}
```

```
boo:
  movss (%rdi), %xmm0
  ret
```

Conversões de Dados


```
double foo (float f) {
  return (double) f;
}
```

```
foo:
  cvtss2sd %xmm0,%xmm0
  ret
```

```
float boo (double *a) {
  return (float)*a;
}
```

```
boo:
  cvtsd2ss (%rdi),%xmm0
  ret
```

Conversões de Dados

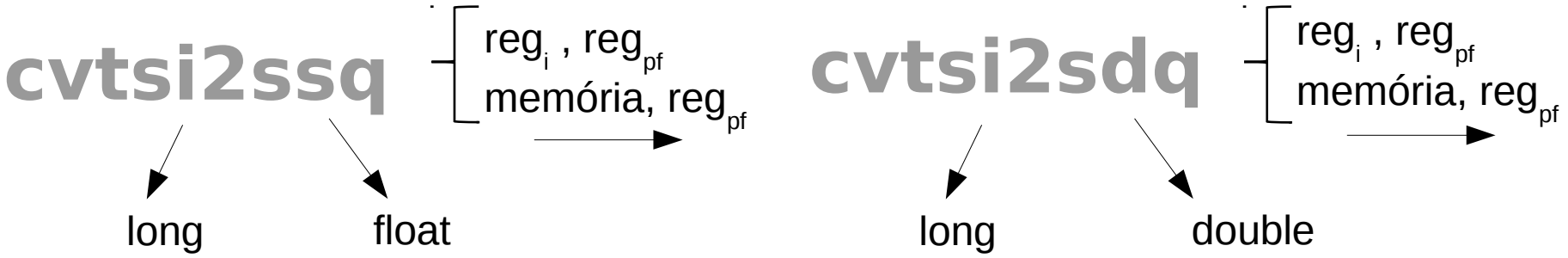

```
double foo (int i) {
  return (double) i;
}
```

```
foo:
  cvtsi2sd %edi,%xmm0
  ret
```

```
float boo (int *a) {
  return (float)*a;
}
```

```
boo:
  cvtsi2ss (%rdi),%xmm0
  ret
```

Conversões de Dados




```
double foo (long i) {
  return (double) i;
}
```

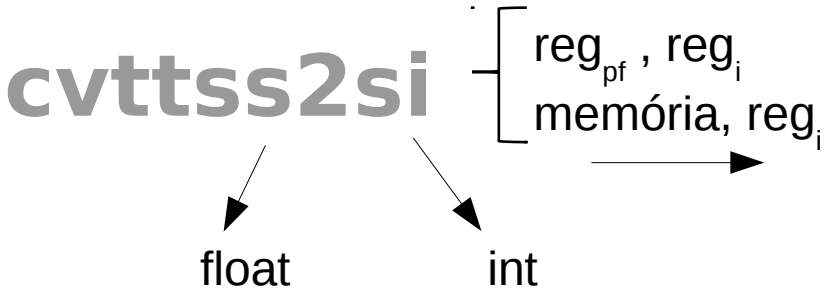
```
foo:
  cvtsi2sdq %rdi,%xmm0
  ret
```

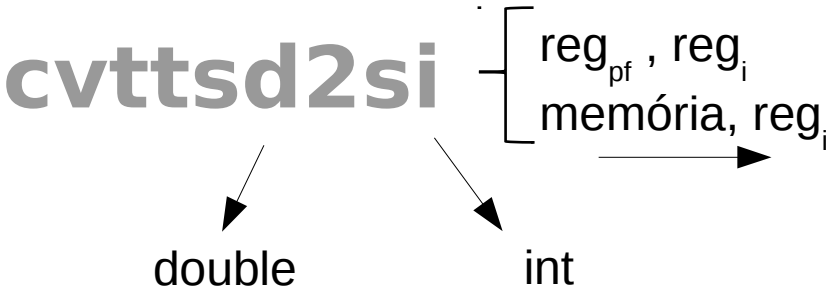
```
float boo (long *a) {
  return (float)*a;
}
```

```
boo:
  cvtsi2ssq (%rdi),%xmm0
  ret
```

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

Conversões com Truncamento





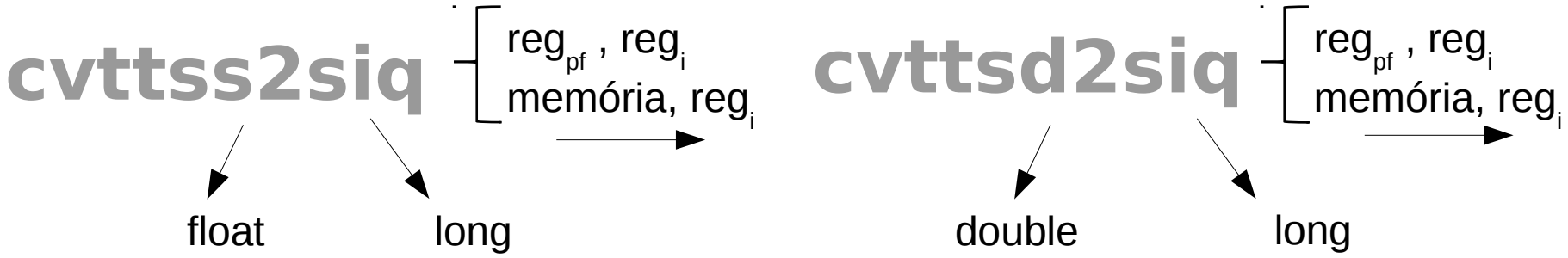

```
int foo (double d) {
  return (int) d;
}
```

```
foo:
  cvttsd2si %xmm0, %eax
  ret
```

```
int boo (float *a) {
  return (int)*a;
}
```

```
boo:
  cvttss2si (%rdi),%eax
  ret
```

Conversões com Truncamento




```
long foo (double d) {
  return (long) d;
}
```

```
foo:
  cvttsd2siq %xmm0, %rax
  ret
```

```
long boo (float *a) {
  return (long)*a;
}
```

```
boo:
  cvttss2siq (%rdi),%rax
  ret
```

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

Operações Aritméticas

adds [s | d]
subs [s | d]
muls [s | d]
divs [s | d]
maxs [s | d]
mins [s | d]
sqrts [s | d]

Exemplo

```
                %xmm0     %xmm1     %xmm2     %edi
                  ↓         ↓         ↓         ↓
double foo (double a, float x, double b, int i) {
  return a*x - b/i;
}
```

```
foo:
  cvtss2sd %xmm1,%xmm1      /* (double) x */
  mulsd %xmm1, %xmm0        /* %xmm0 = a * x */
  cvtsi2sd %edi, %xmm1      /* (double) i */
  divsd %xmm1, %xmm2        /* %xmm2 = b / i */
  subsd %xmm2, %xmm0        /* %xmm0 = a*x - b/i */
  ret
```

Comparação de Valores

ucomiss
ucomisd

- reg_{pf} , reg_{pf}
- memória, reg_{pf}

compara

Testar resultado com as condições de comparações sem sinal (**a**, **ae**, **b**, **be**)

```
int foo (double a, double b) {
  if (a > b) return 1;
  return 0;
}
```

```
foo:
  movl $0, %eax
  ucomisd %xmm1, %xmm0
  jbe fim
  movl $1, %eax
fim:
  ret
```

Preservando Valores

```
double boo(double x);
double foo (double a, double b) {
  return a + boo(b);
}
```

```
foo:
  pushq %rbp
  movq  %rsp, %rbp
  subq  $16, %rsp          /* espaço na pilha para a */
  movsd %xmm0, -8(%rbp)    /* guarda a */
  movsd %xmm1, %xmm0       /* parametro b para boo */
  call  boo                /* retorno em %xmm0 */
  addsd -8(%rbp), %xmm0    /* soma a */
  leave
  ret
```