Operações em vírgula flutuante

João Canas Ferreira

Abril 2019

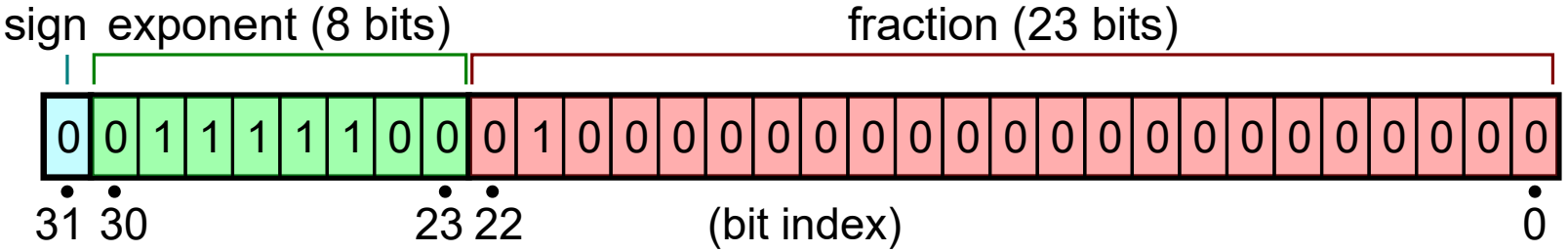
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Assuntos

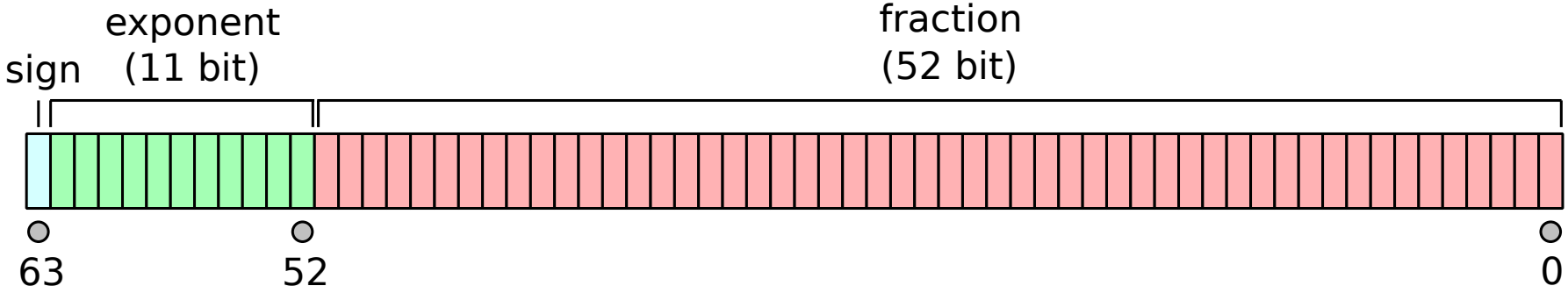
Tipos de dados

A arquitetura AArch64 suporta 3 tipos de dados em vírgula flutuante de acordo com a norma IEEE-754-2008:

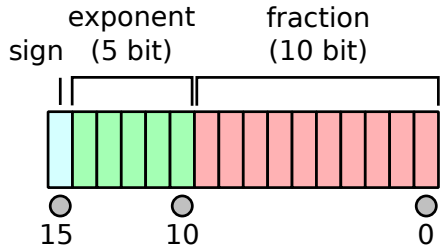
1. precisão simples (`float` em C/C++) — 32 bits



2. precisão dupla (`double` em C/C++) — 64 bits



3. meia precisão (extensão em C; tipo `_Float16`) — 16 bits



Exemplo em C

- Atenção: **Em C**, `_Float16` serve apenas para armazenamento; para cálculos é convertido em `float` ou `double`.

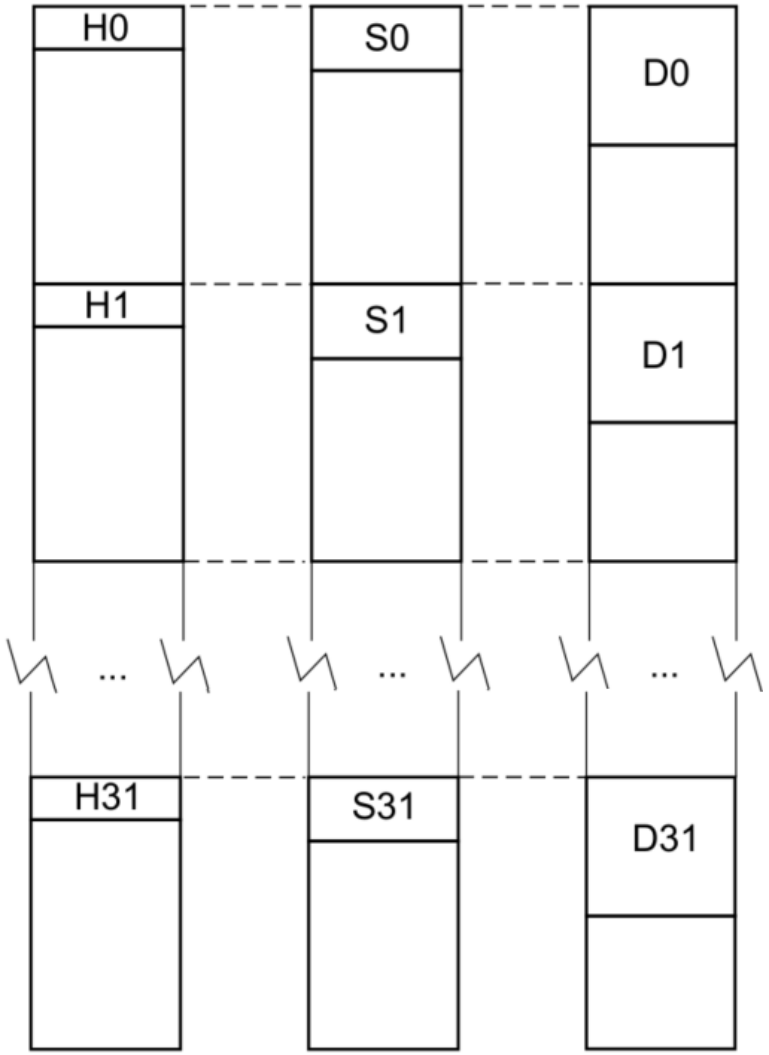
```
#include <stdio.h>
float    vFloat =  126.2346f;
double   vDouble = 124.23434556;
_Float16 vHalf[2];

int main (void)
{
    printf("%.8f %.8f\n", vFloat, vDouble);
    vHalf[0] = vFloat;   vHalf[1] = vDouble;
    vFloat   = vHalf[0];  vDouble = vHalf[1];
    printf("%.8f %.8f\n", vFloat, vDouble);
}
```

- Resultado impresso

```
126.23459625   124.23434556
126.25000000   124.25000000
```

Registos dedicados



- Registos D0–D31: precisão dupla
- Registos S0–S31: precisão simples
  $\rightarrow$ 32 bits menos significativos de D0–D31
- Registos H0–H31: meia precisão
  $\rightarrow$ 16 bits menos significativos de D0–D31 (e de S0–S31)

Resultados e argumentos de sub-rotinas

- Os registos D0-D7 (S0-S7) são usados para passar argumentos e retornar valores (D0 ou S0).
- Registos são atribuídos aos parâmetros por ordem.
  - Por exemplo, se o 1º argumento for do tipo `float` e o 2º do tipo `double`, são passados em `S0` e `D1`, respetivamente.
  - A atribuição de registos VF e de registos inteiros (Xn ou Wn) são independentes.
- Registos 8–15 (S ou D) devem ser preservados pela sub-rotina; os outros registos não são preservados (em geral).

Assuntos

Operações aritméticas

- Operandos são do mesmo tipo, que é também o tipo do resultado
- Formato geral: FXXXX Rdest, Rn, Rm R $\in$ {H,S,D}
- Operações combinadas: FXXXX Rdest, Rn,Rm,Ra R $\in$ {H,S,D}

Instrução	Operação
FADD	Rdest = Rn+Rm
FSUB	Rdest = Rn-Rm
FDIV	Rdest = Rn/Rm
FMUL	Rdest = Rn × Rm
FNMUL	Rdest = -(Rn × Rm)
FMADD	Rdest = (Rn × Rm) + Ra
FNMADD	Rdest = - (Rn × Rm) - Ra
FMSUB	Rdest = - (Rn × Rm) + Ra
FNMSUB	Rdest = (Rn × Rm) - Ra

Funções matemáticas

- Operandos são do mesmo tipo, que é também o tipo do resultado
- Formato geral (1 operando): FXXXX Rdest, Rn R $\in$ {H,S,D}
- Formato geral (2 operandos): FXXXX Rdest, Rn, Rm R $\in$ {H,S,D}

Instrução	Operação
FABS	Rdest = |Rn|
FMAX	Rdest = max(Rn;Rm)
FMIN	Rdest = min(Rn;Rm)
FNEG	Rdest = -Rn
FSQRT	Rdest = $\sqrt{Rn}$
FRINTI	Rdest = arredondar(Rn)

Movimentação de dados

- Entre registos VF, sem conversão (FMOV)
  - FMOV Rd, Rn Rd $\leftarrow$ Rn
- Entre registos VF e registos de uso geral *sem conversão* (FMOV)
  - Wd $\leftarrow$ {Hn, Sn}
  - Xd $\leftarrow$ {Hn, Dn}
  - Hd $\leftarrow$ {Wn, Xn}
  - Sd $\leftarrow$ Wn
  - Dd $\leftarrow$ Xn
- Entre registos e memória: LDR, LDUR, STR, STUR, LDP e STP
- Condicional: `FCSEL Rd,Rn,Rm,cond`

  Se (cond=true) Rd $\leftarrow$ Rn, senão Rd $\leftarrow$ Rm

Exemplo de operações básicas em VF

```c
#include <stdio.h>
#include <math.h>
float VarF = 34.56f;
double VarG = -M_PI;
double vect[]={1.0, -1.23, 7.56};
extern double vf_func(double *v,
     int n, double coef);

int main(void) {
 const int n = 3;
 double res;
 extern double myVar1;
 res = vf_func(vect,n, 2.5);
 for (int i = 0; i<n; i++)
   printf("%f ",vect[i]);
 printf("\nres=%f VarF=%f VarG=%f
      myVar1=%f\n",
       res, VarF, VarG, myVar1);
 return 0;
}
```

```
2.500000   -3.075000   7.560000
res=3.560000   VarF=34.560001
VarG=1.780000   myVar1=1.780000
```

```
.data
.global myVar1
myVar1: .double 1.78
.extern VarG

.text
.global vf_func
.type vf_func, %function
// X0:  ponteiro para vetor
// w1:  número de elementos
// D0:  coeficiente
vf_func:
   ldr     X10, =vect
   ldr     D1, [x10]
   fmov    D2, D0    // coeficiente em D0
   fmul    D0, D1, D2
   str     D0, [x10]
   ldr     D1, [X10, 8]
   fmul    D0, D1, D2
   str     D0, [x10, 8]
   ldr     D2, myVar1
   ldr     X12, =VarG
   str     D2, [X12]
   fadd    D0, D2, D2
   ret
```

Operações de comparação

- As instruções de comparação afetam os indicadores N, Z, C e V.
- Se os operandos não puderem ser comparados, então N=0, Z=0, C=1 e V=1.
- Todas as operações de VF podem afetar os indicadores.
- Os indicadores podem ser acedidos via registo especial NZCV (onde ocupam os bits 31:28)

Instrução	Operação
FCMP Rn, RM	NZCV=comparação(Rn; Rm)
FCMP Rn, #0.0	NZCV=comparação(Rn; Rm)
FCCMP Rn, Rm, #nzcv, cond	se cond NZCV=comparação(Rn; Rm) senão NZVC=#nzvc

#nzvc: valor entre 0-15, composto pelos quatro bits de NZCV

Desvio: Manipulação direta dos indicadores

- Exemplo de manipulação direta dos indicadores

```
MRS     x1, NZCV

MOV     x2, 0x30000000

BIC     x1, x1, x2 // C e V colocados a 0 (bits 29 e 28)

ORR     x1, x2, 0xC0000000  // N e Z colocados a 1 (bits 31 e 30)

MSR     NZCV, x1  // atualizar indicadores
```

Conversão entre formatos VF

- Instruções para converter entre formatos de precisão diferente.

Instrução	Operação
FCVT Sd, Hn	meia precisão para precisão simples
FCVT Dd, Hn	meia precisão para precisão dupla
FCVT Hd, Sn	precisão simples para meia precisão
FCVT Dd, Sn	precisão simples para precisão dupla
FCVT Hd, Dn	dupla precisão para meia precisão
FCVT Sd, Dn	dupla precisão para precisão simples

- Conversão de formato de precisão mais baixa para precisão mais alta: o valor numérico não é afetado
- Conversão de formato de precisão mais alta para precisão mais baixa: pode ocorrer arredondamento ou produzir NaN.

Conversão VF → inteiros

- A conversão pode gerar uma exceção, se o valor não for representável no formato de destino.
- Para números inteiros em complemento para 2 [com sinal] (R ∈ {H;S;D}):

`FCVTNS Wd, Rn`	arredondar para inteiro 32 bits $[-2^{31}; 2^{31} - 1]$
`FCVNTS Xd, Rn`	arredondar para inteiro 64 bits $[-2^{63}; 2^{63} - 1]$

- Para números inteiros sem sinal (R ∈ {H;S;D}):

`FCVTNU Wd, Rn`	arredondar para inteiro 32 bits $[0; 2^{32} - 1]$
`FCVNTU Xd, Rn`	arredondar para inteiro 64 bits $[0; 2^{64} - 1]$

Conversão inteiros → VF

- A conversão pode gerar uma exceção, se o valor não for representável no formato de destino.
- De números inteiros em complemento para 2 [com sinal] (R ∈ {H;S;D}):

`SCVTF Rd, Wn`	converter inteiro 32 bits para VF
`SCVTF Rd, Xn`	convereter inteiro 64 bits para VF

- De números inteiros sem sinal [binário simples] (R ∈ {H;S;D}):

`UCVTF Wd, Rn`	converter inteiro 32 bits (binário simples) para VF
`UCVTF Xd, Rn`	converter inteiro 64 bits (binário simples) para VF