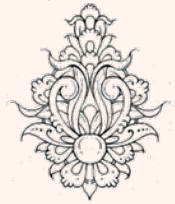
ممیز شناور

زبان ماشین و اسمبلی
(۱۳-۱۱-۰۰۵)

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۴
احمد محمودی ازناوه

http://faculties.sbu.ac.ir/~a_mahmoudi/

فهرست مطالب

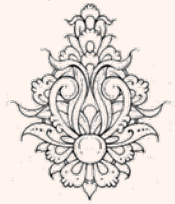


مثال

خروجی برنامه‌ی زیر چیست؟

```c
#include <stdio.h>
int main(){
   float *a;
   int b=0x7F8FFFFF;
   a=(float*)(&b);
   printf("a=%f\n",*a);
}
```

```
ahmad@ubuntu:~/Courses/Assembly/chapter6$ ./a.out
a=nan
```



ممیز شناور

- برای نمایش اعداد اعشاری و اعداد بسیار بزرگ از سیستم عددی میز شناور استفاده می شود.
  - ۳.۱۴۱۵۹۲۶۵
  - ۲.۷۱۸۲۸
  - ۰۰۰۰۰۰۰۰۱ = ۰.۱×۱۰^{-۹}

Copyright 2004 Koren

	IBM/370	DEC/VAX	Cyber 70
Word length (double)	32 (64) bits	32 (64) bits	60 bits
Significand+{hidden bit}	24 (56) bits	$23+1$ $(55+1)$ bits	48 bits
Exponent	7 bits	8 bits	11 bits
Bias	64	128	1024
Base	16	2	2
Range of M	$\frac{1}{16} \le M < 1$	$\frac{1}{2} \le M < 1$	$1 \le M < 2$
Representation of M	Signed-magnitude	Signed-magnitude	One's complement
Approximate range	$16^{63} \approx 7 \cdot 10^{75}$	$2^{127} \approx 1.9 \cdot 10^{38}$	$2^{1023} \approx 10^{307}$
Approximate resolution	$2^{-24} \approx 10^{-7}$ (10^{-17})	$2^{-24} \approx 10^{-7}$ (10^{-17})	$2^{-48} \approx 10^{-14}$

ممیز شناور (ادامه...)

- در سال ۱۹۸۵ استاندارد IEEE Std 754 مطرح شد.
- این استاندارد واگرایی شیوه‌های به کار رفته برای نمایش ممیز شناور را کاهش داد.
  - بدین‌ترتیب برنامه‌های نوشته شده برای مقاصد علمی قابل حمل شدند.
- بر طبق این استاندارد، اعداد به سه شیوه نشان داده می‌شود:

- half precision
- single precision
- double precision

half: 5 bits
single: 8 bits
double: 11 bits

half: 10 bits
single: 23 bits
double: 52 bits

S	Exponent	Fraction

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

Single: Bias = 127; Double: Bias = 1023

Denormal Numbers

اعداد نامنجار

Exponent = 000...0 $\Rightarrow$ hidden bit is 0

$$x = (-1)^S \times (0 + \text{Fraction}) \times 2^{-\text{Bias}}$$

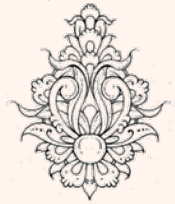
- بدین ترتیب می‌توان اعداد کوچک‌تری را نیز نمایش داد.
- در صورتی که بخش کسری را برابر صفر قرار دهیم:

$$x = (-1)^S \times (0 + 0) \times 2^{-\text{Bias}} = \pm 0.0$$

بدین ترتیب دو نمایش برای **0** خواهیم داشت

ناعدد و بی‌نهایت

- Exponent = 111...1, Fraction = 000...0
  - $\pm\infty$
    - در محاسبات بعدی نیز قابل استفاده است.
- Exponent = 111...1, Fraction ≠ 000...0
    - ناعدد (Not-a-Number (NaN))
    - بیان‌گر محاسبات نادرست می‌باشد.
    - این اعداد نیز قابلیت استفاده در محاسبات بعدی را دارند.



مثال

```
ahmad@ubuntu:~/Courses/Assembly/chapter6$ gcc denor.c -std=c99
ahmad@ubuntu:~/Courses/Assembly/chapter6$ time ./a.out
*a=0.00000000000000000000000000000000001175494490952
sum=0.00000000000000000000000000000039443045261050590271

real    0m10.527s
user    0m10.429s
sys     0m0.008s
```

```c
#include <stdio.h>
#include <limits.h>
int main(){
  float *a;
  int b=0x00000001;

  a=(float*)(&b);
  printf("*a=%.50f\n",*a);

 float sum=0;
 for(int i=0; i<INT_MAX;i++)
    sum=sum+(*a);
  printf("sum=%.50f\n",sum);
}
```

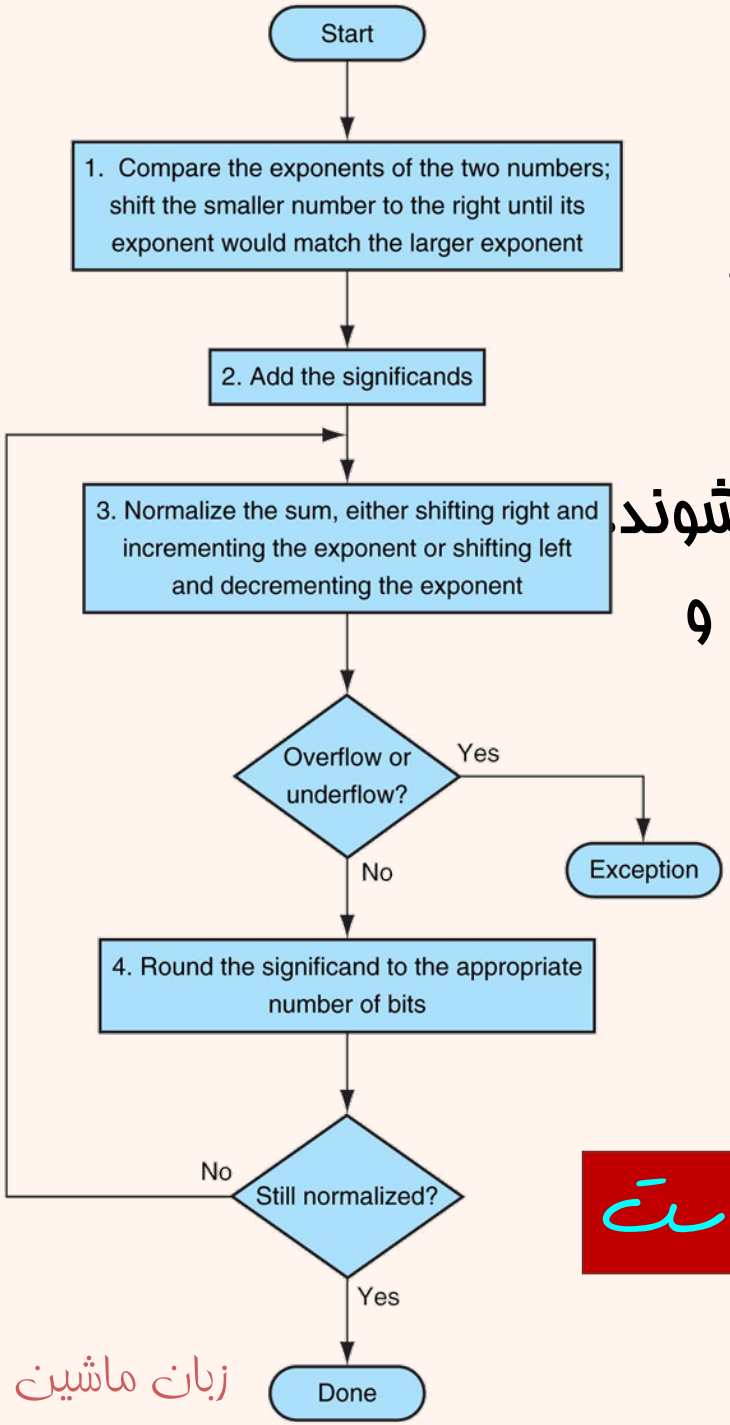
```c
#include <limits.h>
int main(){
  float *a;
  int b=0x00800001;

  a=(float*)(&b);
  printf("*a=%.50f\n",*a);

 float sum=0;
 for(int i=0; i<INT_MAX;i++)
    sum=sum+(*a);
  printf("sum=%.50f\n",sum);
}
```

```
ahmad@ubuntu:~/Courses/Assembly/chapter6$ gcc denor.c -std=c99
ahmad@ubuntu:~/Courses/Assembly/chapter6$ time ./a.out
*a=0.00000000000000000000000000000000000000000140130
sum=0.00000000000000000000000000000000000002350988701645

real    5m5.527s
user    4m59.323s
sys     0m0.288s
```

مشخصات اعداد ممیز شناور IEEE 754

Feature	Single/Short	Double/Long
Word width in bits	32	64
Significand in bits	23 + 1 hidden	52 + 1 hidden
Significand range	$[1, 2 - 2^{-23}]$	$[1, 2 - 2^{-52}]$
Exponent bits	8	11
Exponent bias	127	1023
Zero (±0)	e + bias = 0, f = 0	e + bias = 0, f = 0
Denormal	e + bias = 0, $f \neq 0$ represents $\pm 0.f \times 2^{-126}$	e + bias = 0, $f \neq 0$ represents $\pm 0.f \times 2^{-1022}$
Infinity (±∞)	e + bias = 255, f = 0	e + bias = 2047, f = 0
Not-a-number (NaN)	e + bias = 255, $f \neq 0$	e + bias = 2047, $f \neq 0$
Ordinary number	e + bias $\in [1, 254]$ $e \in [-126, 127]$ represents $1.f \times 2^e$	e + bias $\in [1, 2046]$ $e \in [-1022, 1023]$ represents $1.f \times 2^e$
min	$2^{-126} \cong 1.2 \times 10^{-38}$	$2^{-1022} \cong 2.2 \times 10^{-308}$
max	$\cong 2^{128} \cong 3.4 \times 10^{38}$	$\cong 2^{1024} \cong 1.8 \times 10^{308}$

جمع در ممیز شناور



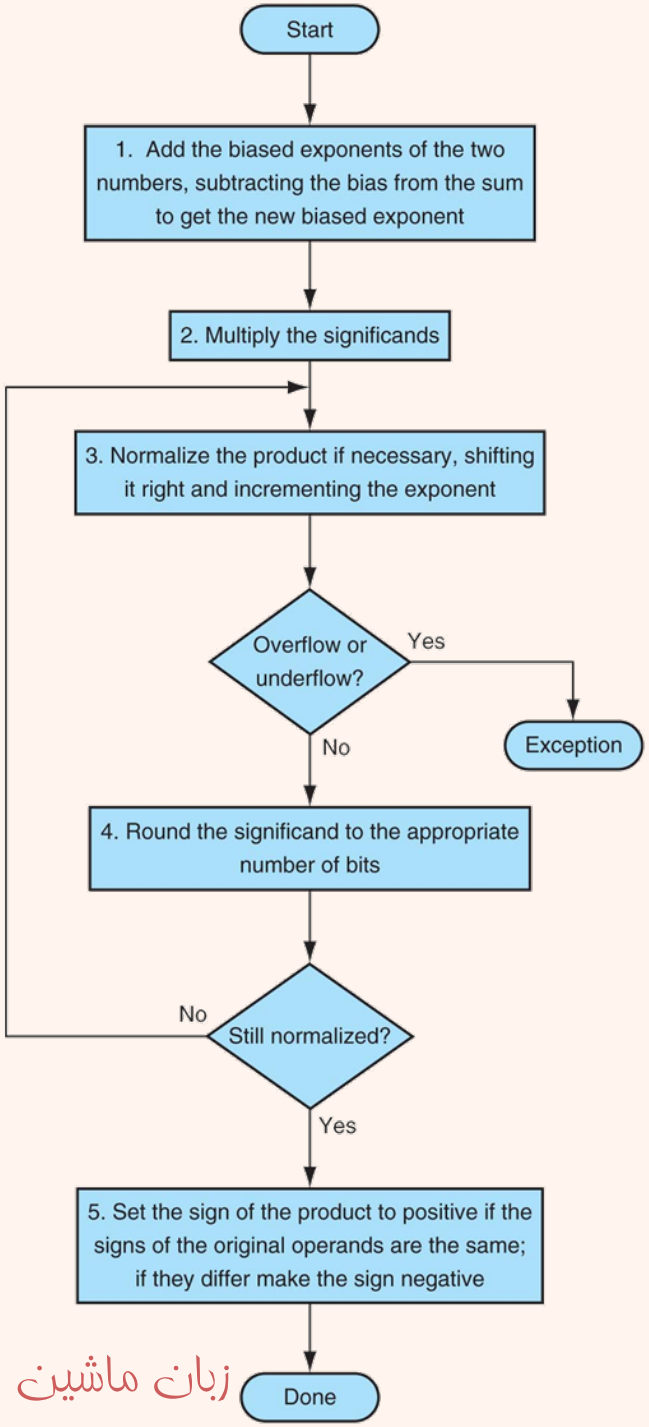
1. ابتدا توان‌ها را یکسان می‌کنیم.
   - این کار با شیفت عدد کوچک‌تر به راست انجام می‌شود.
2. مقادیر اعشاری با هم جمع می‌شوند.
3. حاصل بهنجار شده و وقوع سرریز و فروریز بررسی می‌شود.
4. حاصل گرد می‌شود.

- مجددا بهنجار بودن عدد بررسی می‌شود.

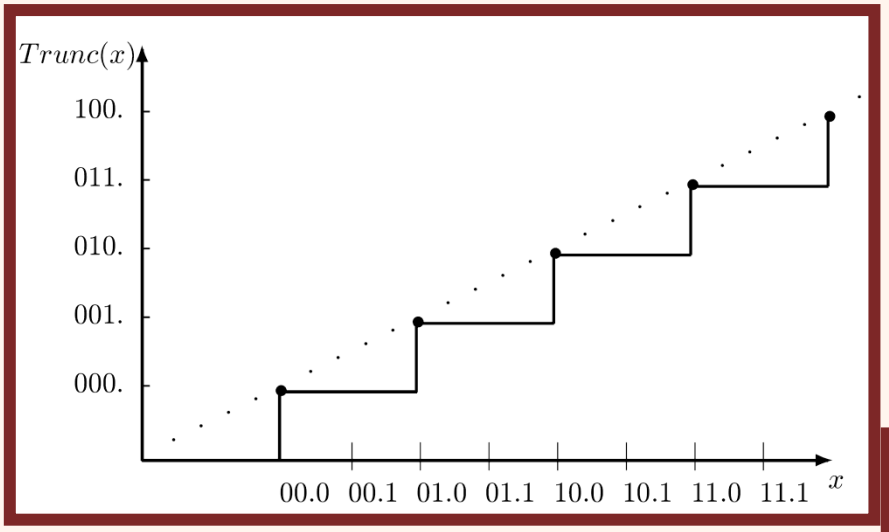
نسبت به اعداد صحیح پیچیده‌تر است

ضرب ممیز شناور

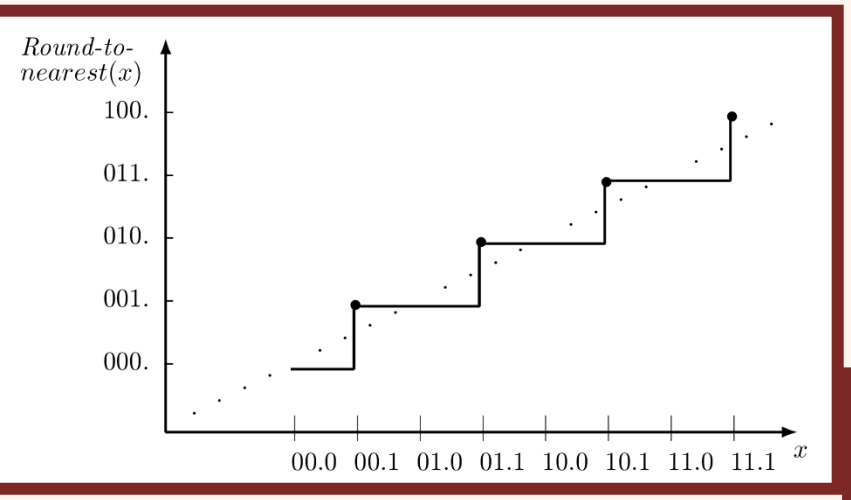
1. توان‌ها با هم جمع می‌شود
2. significandها در هم ضرب می‌شوند.
3. اعداد بهنجار شده و بروز سرریز یا فروریز چک می‌شود.
4. اعداد گرد می‌شوند و در صورت نیاز مجدد بهنجار می‌شوند.
5. علامت عدد تعیین می‌شود.



گرد کردن



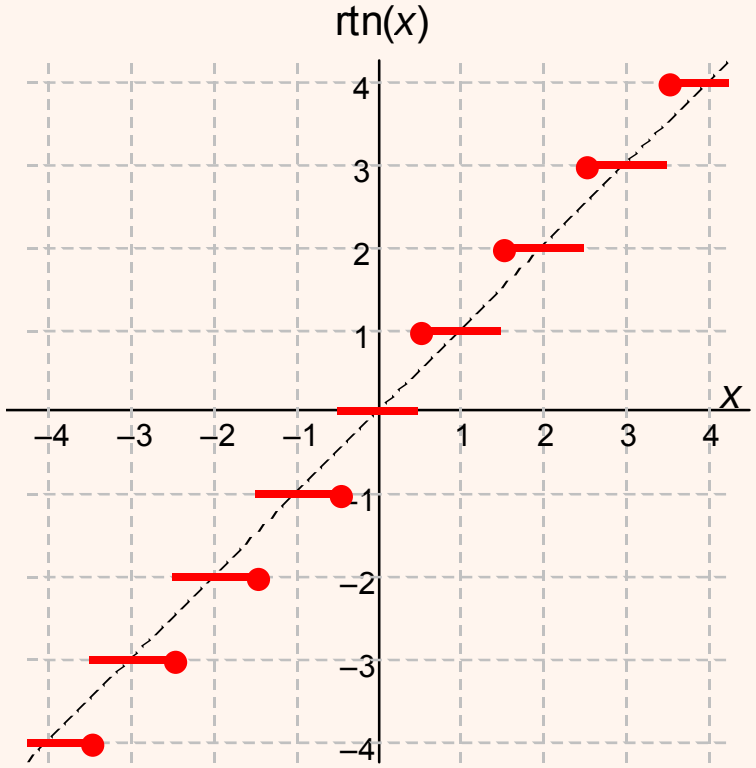
Number	$Trunc(x)$	Error
$X.00$	X	0
$X.01$	X	$-1/4$
$X.10$	X	$-1/2$
$X.11$	X	$-3/4$

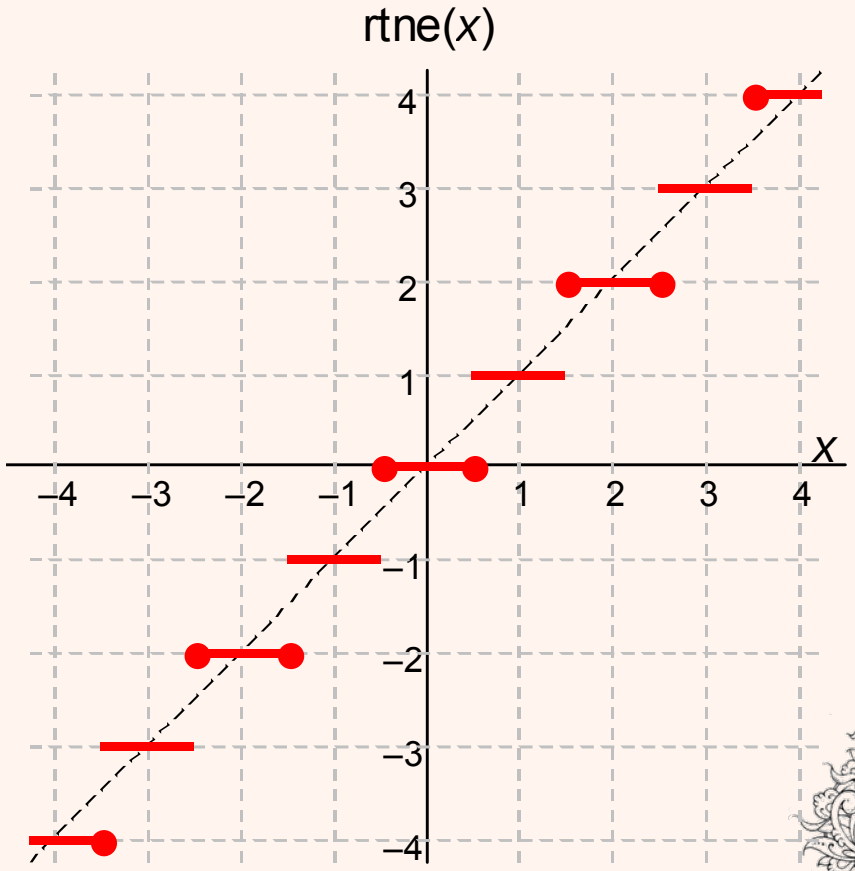


Number	*Round-to-nearest(x)*	Error
$X.00$	X	0
$X.01$	X	$-1/4$
$X.10$	$X + 1$	$+1/2$
$X.11$	$X + 1$	$+1/4$

زبان ماشین
Copyright 2004 Koren

گرد کردن به نزدیک‌ترین مقدار زوج





Number	$Round(x)$	Error	Number	$Round(x)$	Error
$X0.00$	$X0.$	0	$X1.00$	$X1.$	0
$X0.01$	$X0.$	$-1/4$	$X1.01$	$X1.$	$-1/4$
$X0.10$	$X0.$	$-1/2$	$X1.10$	$X1.+1$	$+1/2$
$X0.11$	$X1.$	$+1/4$	$X1.11$	$X1.+1$	$+1/4$

زبان ماشین

کم‌ارزش‌ترین بیت قابل ذخیره‌سازی(ulp)

...کردن در استاندارد IEEE...

بیت‌هایی که خارج از محدوده هستند و در جریان محاسبات به وجود می‌آیند.

LSB	R	S	Operation	$\overline{Error}$
0	0	0	+ 0	0
0	0	1	+ 0	−0.25 *ulp*
0	1	0	+ 0	−0.50 *ulp*
0	1	1	+0.5 *ulp*	+0.25 *ulp*
1	0	0	+ 0	0
1	0	1	+ 0	−0.25 *ulp*
1	1	0	+0.5 *ulp*	+0.50 *ulp*
1	1	1	+0.5 *ulp*	+0.25 *ulp*
			Total	0

(a) Round-to-nearest-even scheme

R	S	Operation	$\overline{Error}$
0	0	+ 0	0
0	1	+ 0	−0.25 *ulp*
1	0	+ 0	−0.50 *ulp*
1	1	+ 0	−0.75 *ulp*
		Total	−0.375 *ulp*

(b) Round-to-zero scheme

Sign	R	S	Operation
+	0	0	+ 0
+	0	1	+1 *ulp*
+	1	0	+1 *ulp*
+	1	1	+1 *ulp*
−	0	0	+ 0
−	0	1	+ 0
−	1	0	+ 0
−	1	1	+ 0

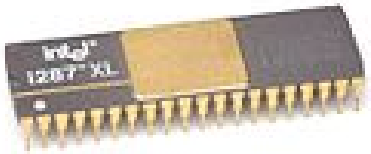
(c) Round-to-plus-infinity scheme

Sign	R	S	Operation
−	0	0	+ 0
−	0	1	+1 *ulp*
−	1	0	+1 *ulp*
−	1	1	+1 *ulp*
+	0	0	+ 0
+	0	1	+ 0
+	1	0	+ 0
+	1	1	+ 0

(d) Round-to-minus-infinity scheme

Copyright 2004 Koren

عملیات ممیز شناور

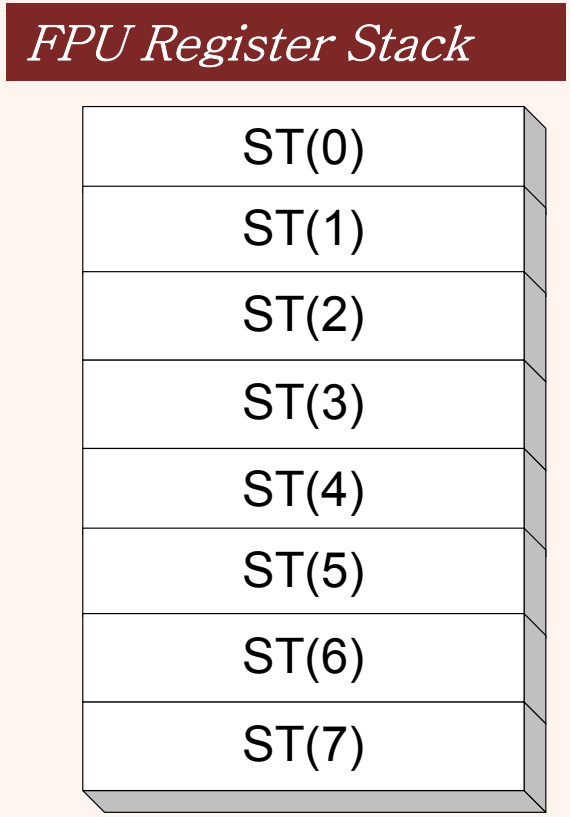


Intel 80287XL

- نخستین پردازنده‌ای که از استاندارد IEEE754 حمایت می‌کرد، ۸۰۸۶ بود که از یک کمک‌پردازنده با نام ۸۰۸۷ FPU استفاده می‌کرد، چرا که صنعت نیمه‌هادی آن‌قدر پیشرفت نکرده بود که واحد ممیز شناور را در کنار پردازنده‌ی اصلی قرار دهد.
- در پردازنده‌هایی که پیش از ۸۰۴۸۶ ارائه شده بودند، برای انجام عملیات ممیز شناور از طریق نرم‌افزار و با استفاده از دستورالعمل‌های صحیح آن را شبیه‌سازی می‌کردند، یا این که کمک‌پردازنده‌ی عملیات ممیز شناور جداگانه خریداری می‌شده است.

ثبات‌های ممیزشناور

- واحد ممیزشناور دارای مجموعه‌ی کاملی از ثبات‌هاست.
- این واحد دارای هشت ثبات هشتاد بیتی برای داده‌های ممیزشناور است.
- دسترسی این ثبات‌ها به صورت پشته است.



Data Type	Length	Significand Bits	Exponent Bits	Range	
Single precision	32	24	8	1.18 x 10^-38 to 3.40 x 10^38	C float
Double precision	64	53	11	2.23 x 10^-308 to 1.79 x 10^308	C double
Double extended	80	64	15	3.37 x 10^-4932 to 1.18 x 10^4932	C long double

.float
.double
.tfloat

زبان ماشین

انتقال داده به ثبات‌های ممیز شناور

Floating-Point Load

```
fldx source
```

این دستور برای انتقال داده از حافظه یا ثبات‌های ممیز شناور به ثبات‌های ممیز شناور به کار می‌رود

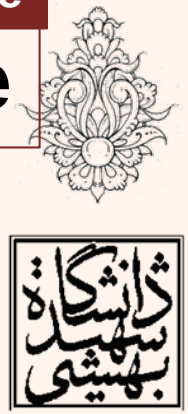
s -- Short (single precision, 32 bits)
l -- Long (64 bits).
t -- Ten-byte (80 bits).

Floating-Point Store

```
fst(p)x source
```

pop

داده از روی پشته خوانده (برداشته) می‌شود یا روی پشته نوشته می‌شود



مثال

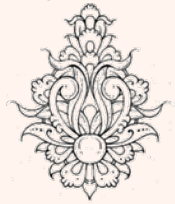
```
# floattest.s - An example of
using floating point numbers
.section .data
value1:
    .float 12.34
value2:
    .double 2353.631
.section .bss
    .lcomm data, 8
.section .text
.globl _start
_start:
    nop
    flds value1
    fldl value2
    fstl data

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
st0             0          (raw 0x00000000000000000000)
st1             0          (raw 0x00000000000000000000)
st2             0          (raw 0x00000000000000000000)
st3             0          (raw 0x00000000000000000000)
```

```
st0             12.340000152587890625    (raw 0x4002c570a40000000000)
st1             0          (raw 0x00000000000000000000)
st2             0          (raw 0x00000000000000000000)
st3             0          (raw 0x00000000000000000000)
st4             0          (raw 0x00000000000000000000)
st5             0          (raw 0x00000000000000000000)
```

```
st0             2353.6309999999998581188265234231949     (raw 0x400a931a189374bc6800)
st1             12.340000152587890625    (raw 0x4002c570a40000000000)
st2             0          (raw 0x00000000000000000000)
st3             0          (raw 0x00000000000000000000)
st4             0          (raw 0x00000000000000000000)
st5             0          (raw 0x00000000000000000000)
st6             0          (raw 0x00000000000000000000)
st7             0          (raw 0x00000000000000000000)
```

```
(gdb) x /gf  &data
0x80491a0 <data>:       2353.6309999999999
(gdb)
```



شکل دستورها

- امکان انتقال داده بین ثبات‌های ممیزشناور و ثبات‌های معمولی وجود ندارد و برای چنین کاری باید از حافظه به عنوان واسطه استفاده کرد. (البته استثنا هم وجود دارد!)
- حرف اول دستورهای ممیزشناور، «f» است، بدین‌ترتیب از دستورهای معمولی متمایز می‌شود.
- حرف دوم، مشخص می‌کند داده‌ی مورد استفاده از چه نوعی است:
  - i نشان‌دهنده‌ی این است که داده مورد استفاده داده‌ی صحیح است.
  - وگرنه داده عددی حقیقی است.

```
fild  - load integer number
fmul – real number multiply
```

مقداردهی ثبات‌های ممیزشناور

```
fld memory/freg (real)
```

یک عدد حقیقی از حافظه/ثبات ممیزشناور خوانده شده در ثبات ممیزشناور قرار می‌گیرد

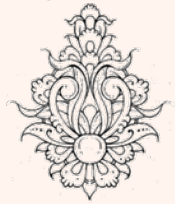
```
fild memory (integer)
```

یک عدد صحیح از حافظه خوانده شده و پس از تبدیل به فرمت ممیزشناور در ثبات ممیزشناور قرار می‌گیرد

- در خانواده‌ی x86 دستورهایی وجود دارند که مقادیر خاصی را در ثبات‌های ممیزشناور قرار می‌دهند:

```
fldxx
```

Floating-Point Load Constants



دستورهای بازگذاری داده در ممیزشناور

fld1

عدد «1» در پشته‌ی ممیزشناور قرار می‌گیرد

fldz

عدد «0» در پشته‌ی ممیزشناور قرار می‌گیرد

fldpi

عدد «π» در پشته‌ی ممیزشناور قرار می‌گیرد

fldl2e

عدد «$Log_2(e)$» در پشته‌ی ممیزشناور قرار می‌گیرد

fldl2t

عدد «$Log_2(10)$» در پشته‌ی ممیزشناور قرار می‌گیرد

fldlg2

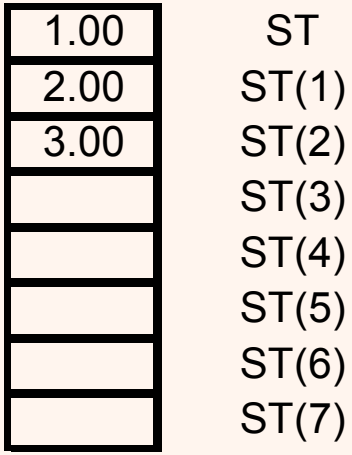
عدد «$Log_{10}(2)$» در پشته‌ی ممیزشناور قرار می‌گیرد

fldln2

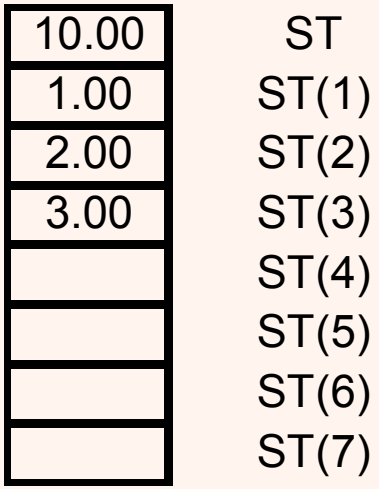
عدد «Ln (2)» در پشته‌ی ممیزشناور قرار می‌گیرد

مثال

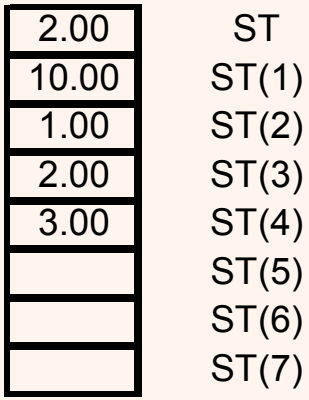
```
.section .data
fpValue:
    .float 10.0
intValue:
    .int 20
intValue2:
  .int 30
```




```
flds fpValue
```




```
fld %st(2)
```



مثال

fild intValue

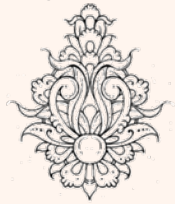
20.00	ST
2.00	ST(1)
10.00	ST(2)
1.00	ST(3)
2.00	ST(4)
3.00	ST(5)
	ST(6)
	ST(7)

fild intValue2

30.00	ST
20.00	ST(1)
2.00	ST(2)
10.00	ST(3)
1.00	ST(4)
2.00	ST(5)
3.00	ST(6)
	ST(7)

fldz

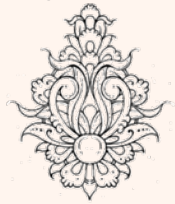
0.00	ST
30.00	ST(1)
20.00	ST(2)
10.00	ST(3)
2.00	ST(4)
1.00	ST(5)
2.00	ST(6)
3.00	ST(7)



مثال

```
# fpuvals.s point
constants
.section .text
.globl _start
_start:
    nop
    fld1
    fldl2t
    fldl2e
    fldpi
    fldlg2
    fldln2
    fldz

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
st0            0        (raw 0x00000000000000000000)
st1            0.693147180559945309428690474184975 3    (raw 0x3ffeb17217f7d1cf79ac)
st2            0.301029995663981195225646428359489 45   (raw 0x3ffd9a209a84fbcff799)
st3            3.141592653589793238512808959406186 2    (raw 0x4000c90fdaa22168c235)
st4            1.442695040888963407387651782798343 4    (raw 0x3fffb8aa3b295c17f0bc)
st5            3.321928094887362347808340556909456 6    (raw 0x4000d49a784bcd1b8afe)
st6            1        (raw 0x3fff8000000000000000)
st7            0        (raw 0x00000000000000000000)
```



دستورهای ذخیره‌سازی در ممیز شناور

```
fst   %st(num)
```

مقدار st را در st(num) ذخیره می‌کند

```
fstp  %st(num)
```

مقدار st را در st(num) ذخیره می‌کند و st از روی پشته برداشته می‌شود

```
fst memory(real)
```

مقدار st در حافظه ذخیره می‌شود

```
fstp memory(real)
```

مقدار st از روی پشته برداشته و حافظه ذخیره می‌شود

```
fist memory(integer)
```

مقدار st ابتدا به مقدار صحیح تبدیل شده و سپس ذخیره می‌شود

```
fistp memory(integer)
```

مانند حالت قبل با این تفاوت که داده از روی پشته برداشته می‌شود

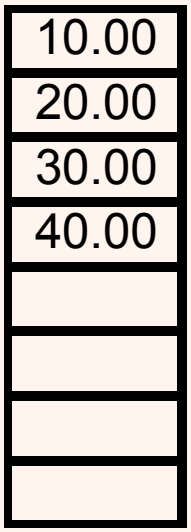
```
fxch
```

محتوای st و st(1) را عوض می‌کند

```
fxch %st(num)
```

محتوای st و st(num) را عوض می‌کند

مثال



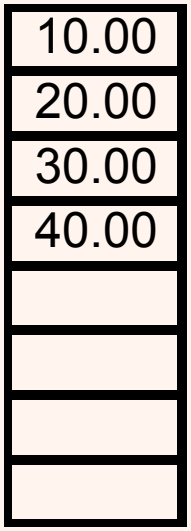
ST
ST(1)
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)
ST(7)

```
intValue:
    .int 0
```

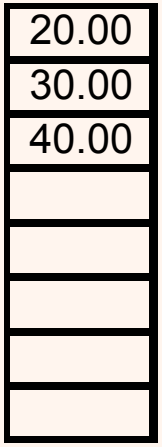
حالت اولیه

fist intvalue

intValue=0000000A



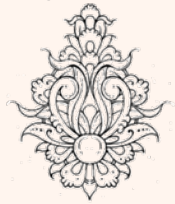
ST
ST(1)
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)
ST(7)



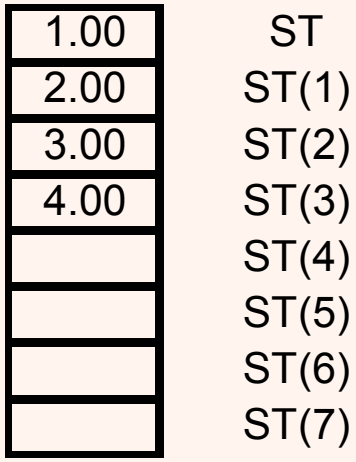
ST
ST(1)
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)
ST(7)

fistp intvalue

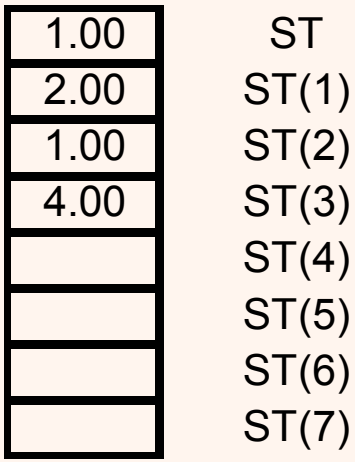
intValue=0000000A



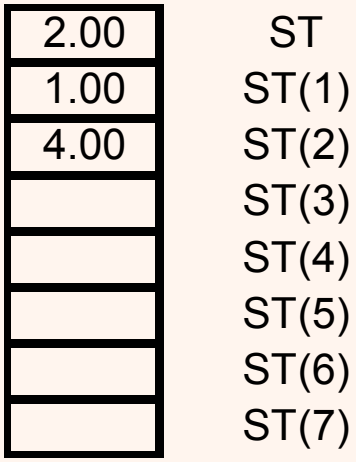
مثال




```
fst %st(2)
```




```
fstp %st(2)
```



واحد ممیزشناور در یک نگاه

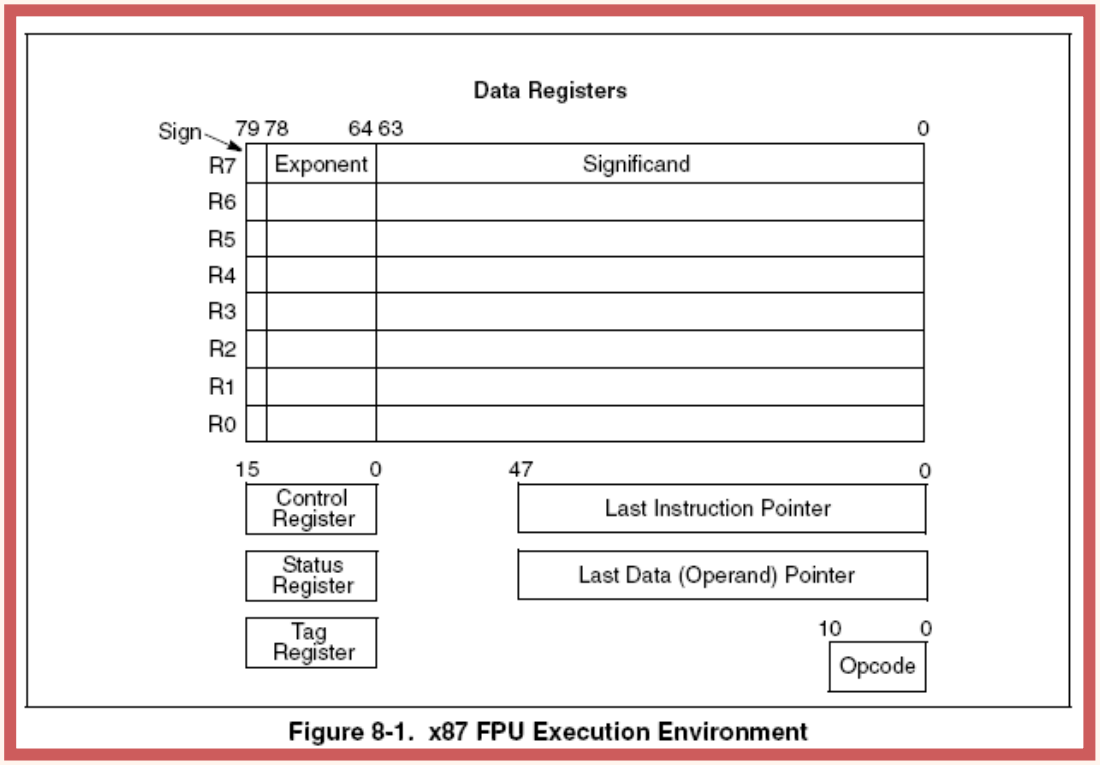
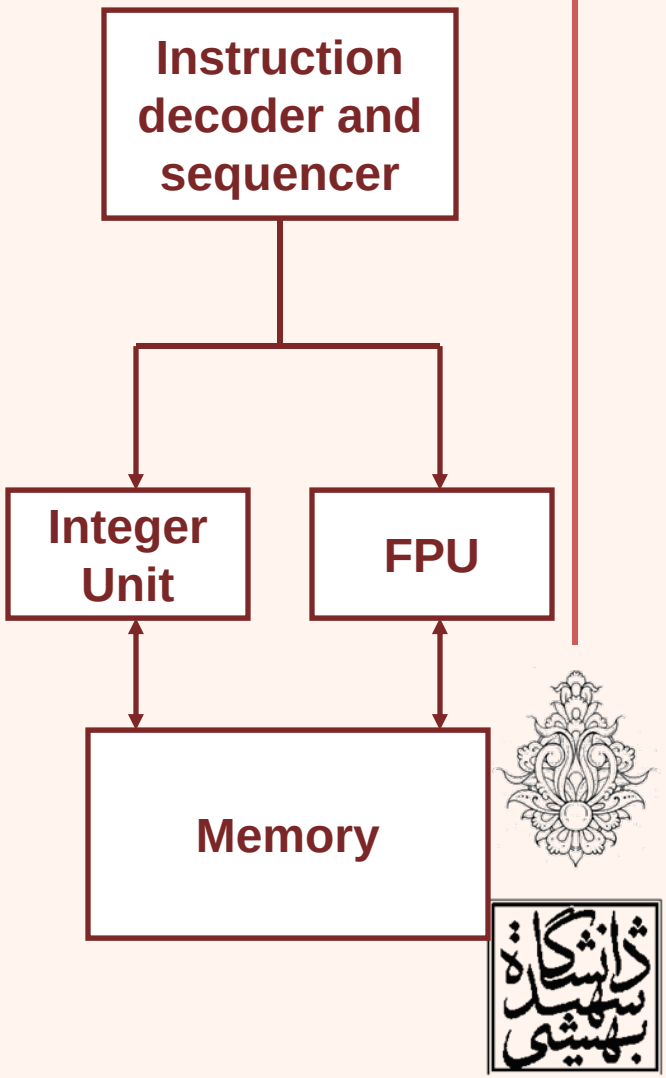
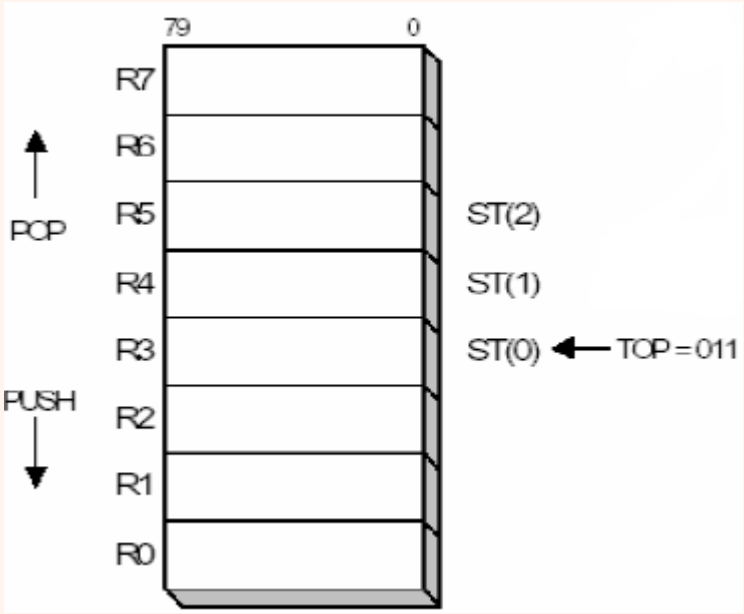


Figure 8-1. x87 FPU Execution Environment



ثبات‌های ممیزشناور

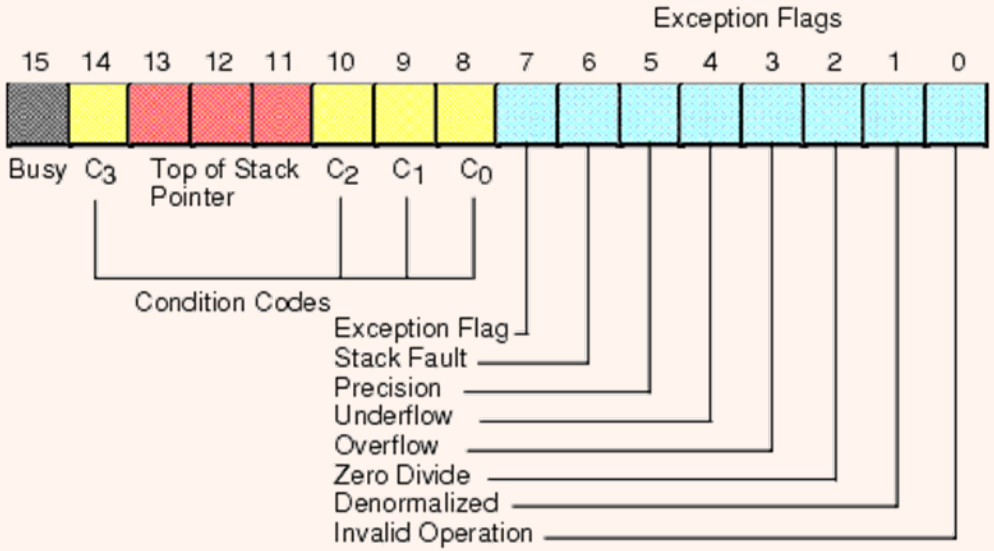
- با توجه به این واحد ممیزشناور مستقل از پردازنده‌ی اصلی است، به ثبات‌های پرچم نیز دسترسی ندارد، از این رو ثبات‌های دیگری نیز برای آن تعریف شده است.
- تا به حال با ثبات داده آشنا شدیم:



ثبات وضعیت

status word register

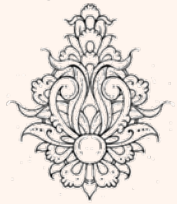
- این ثبات بیان‌گر وضعیت واحد ممیزشناور است.
- بیت‌های ۱۱ تا ۱۳ ثبات روی پشته را مشخص می‌کند، پیش از اضافه شدن داده به پشته مقدار آن یک واحد کاهش می‌یابد.



http://webster.cs.ucr.edu/AoA/Windows/HTML/RealArithmetic.html

ثبات وضعیت(ادامه ...)

- بیت صفر تا شش بروز استثنائات را نشان می‌دهد.
  - بیت صفر: دستورالعمل نامشخص
  - بیت یک: اعداد ناهنجار
  - بیت دو: تقسیم بر صفر
  - بیت‌های سه و چهار: سرریز و فروریز
  - بیت پنج: محاسبات نادقیق
  - بیت ششم : سرریز یا فررویز در پشته، در صورت سرریز C_1 نیز فعال خواهد شد.(یک بودن: overflow)
  - در صورت بروز هر نوع استثنا بیت هفتم نیز فعال می‌شود.
- بیت‌های ۸، ۹، ۱۰ و ۱۴ در دستورهای شرطی به کار می‌روند(C_0 تا C_3).



ثبات وضعیت (ادامه ...)

- محتوای این ثبات را می‌توان به حافظه یا ثبات ax منتقل نمود:

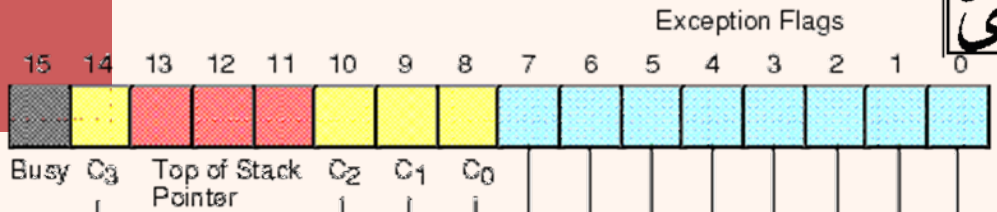
Store Floating-Point Status Word

fstsw

```
.section .bss
   .lcomm status, 2
.section .text
.globl _start
_start:
   nop
   fstsw %ax
   fldpi
   fstsw status

   movl $1, %eax
   movl $0, %ebx
   int $0x80
```

```
fstat           0x0         0
```

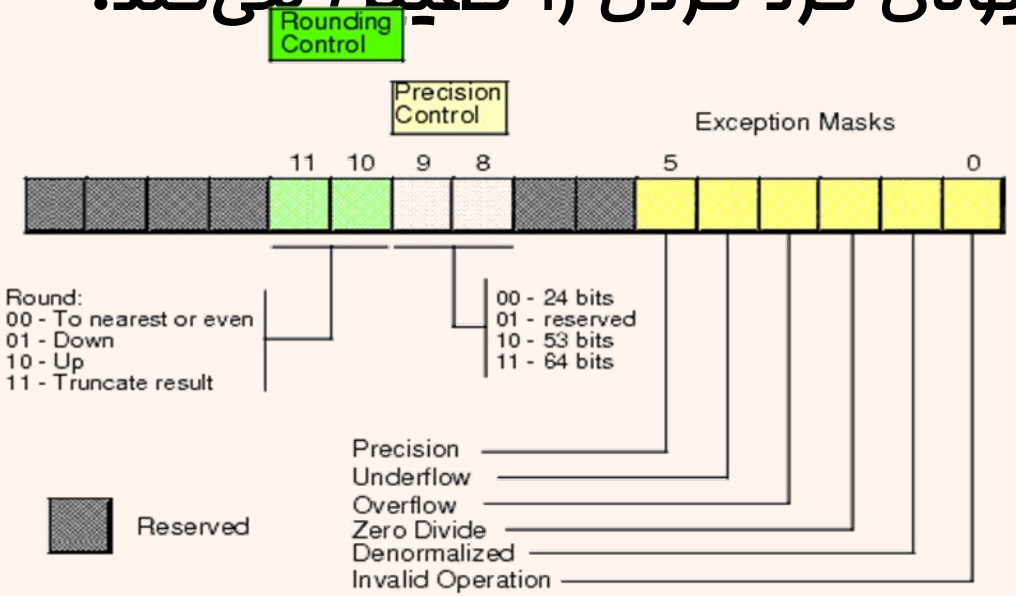
```
(gdb) x &status
0x8049190 <status>:     0x00003800
```



ثبات کنترلی

control word register

- شش بیت کم‌ارزش برای فعال/غیرفعال کردن استثنائات به کار می‌رود. در صورت یک بودن غیرفعال می‌شوند.
- بیت ۸ و ۹ دقت عملیات را مشخص می‌کند.
- بیت ۱۰ و ۱۱ شیوه‌ی گرد کردن را تعیین می‌کند.




```
.section .text
.globl _start
_start:
    nop
```

```
fctrl           0x37f    895
```


http://webster.cs.ucr.edu/AoA/Windows/HTML/RealArithmetic.html

ثبات کنترلی

- محتوای ثبات کنترلی را می‌توان به حافظه منتقل کرد.

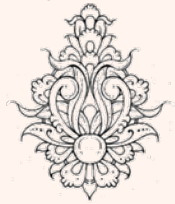
Store Floating-Point Control Word

```
fstcw
```

- همچنین می‌توان محتوای حافظه را به این ثبات منتقل کرد.

Load Floating-Point Control Word

```
fldcw
```

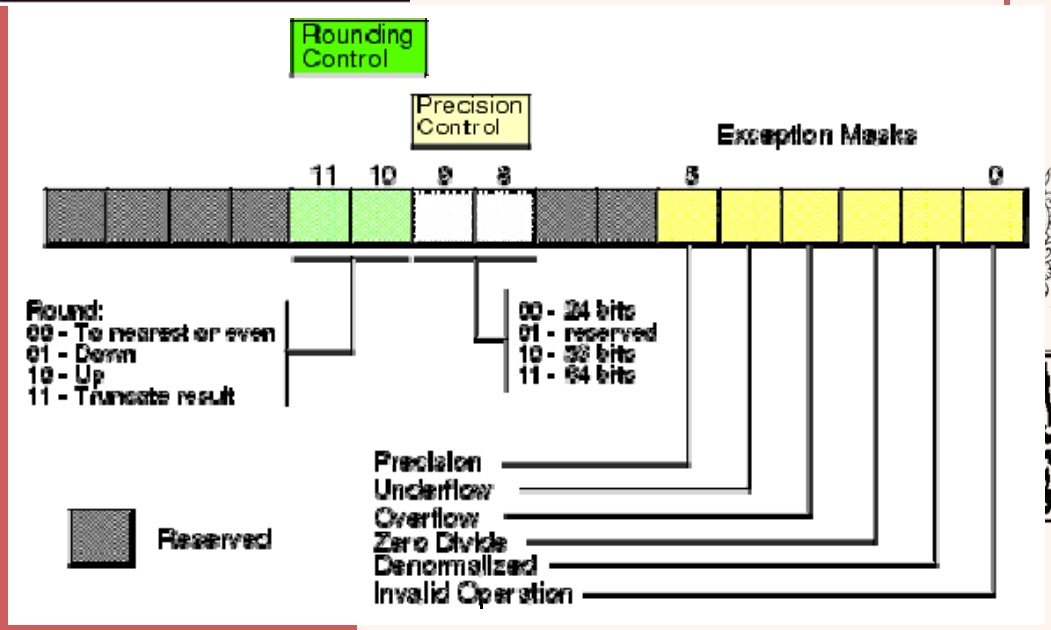


مثال

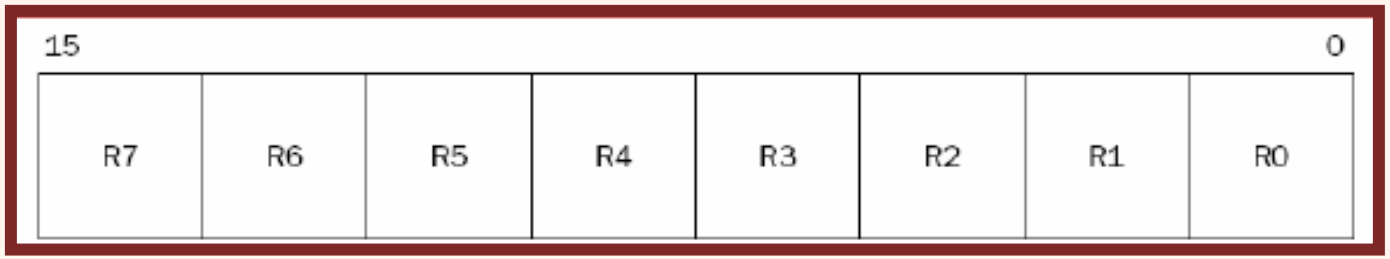
- دقت محاسبات ممیزشناور را به single تغییر دهید.

```
.section .data
newvalue:
    .byte 0x7f, 0x00
.section .bss
    .lcomm control, 2
.section .text
.globl _start
_start:
    nop
    fstcw control
    fldcw newvalue


    fstcw control

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
(gdb) print /x $fctrl
$1 = 0x7f
```



ثبات برچسب



- به هر ثبات دو بیت اختصاص داده شده است.
  - داده‌ی غیر صفر معتبر(00)
  - صفر (01)
  - مقدار خاص (10)
  - خالی (11)

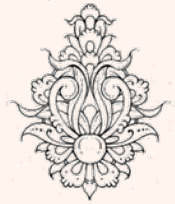
دستور مقداردهی اولیه

finit

Initialize floating point processor

در ابتدای برنامه از آن استفاده می‌شود
محتوای ثبات‌ها را پاک کرده و ثبات st(0) در آن از نو مقدار دهی می‌شود و باعث می‌شود سایر ثبات‌ها به مقدار پیش‌فرض تغییر وضعیت دهند

```
finit
fld1
fldz
fxchg
```



مثال

```
.section .data
value1:
    .int 40
value2:
    .float 92.4405
value3:
    .double 221.440321
.section .bss
    .lcomm int1, 4
    .lcomm control, 2
    .lcomm status, 2
    .lcomm result, 4
.section .text
.globl _start
```

```
_start:
    nop
    finit
    fstcw control
    fstsw status
    fildS value1
    fists int1
    flds value2
    fldl value3
```

```
fctrl           0x37f    895
fstat           0x0      0
ftag            0xffff   65535
```

```
fstat           0x3800   14336
ftag            0x3fff   16383
```

```
(gdb) info float
  R7: Valid    0x4004a000000000000000 +40
=>R6: Valid    0x4006dd70b8e086bdf800 +221.4403210000000115
  R5: Empty    0x4005b8e1890000000000
  R4: Empty    0x00000000000000000000
  R3: Empty    0x00000000000000000000
  R2: Empty    0x00000000000000000000
  R1: Valid    0x4006dd70b8e086bdf800 +221.4403210000000115
  R0: Empty    0x00000000000000000000

Status Word:         0x3000
                       TOP: 6
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x0ff3
Instruction Pointer: 0x73:0x0804812e
Operand Pointer:     0x7b:0x080491cc
Opcode:              0xd91d
```

```
st0             221.4403210000000115087459
st1             40        (raw 0x4004a00000
st2             0         (raw 0x0000000000
st3             221.4403210000000115087459
st4             0         (raw 0x0000000000
st5             0         (raw 0x0000000000
st6             0         (raw 0x0000000000
st7             92.44049835205078125
```

```
fctrl           0x37f    895
fstat           0x3000   12288
ftag            0xff3    4083
```

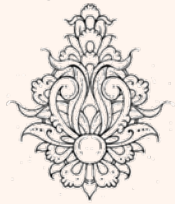
```
(gdb) x /d  &int1
0x80491c4 <int1>:       40
(gdb) x /f  &result
0x80491cc <result>:     92.4404984
(gdb)
```


زبان ماشین

عملیات پایه‌ی ممیز شناور

Instruction	Description
FADD	Floating-point addition
FDIV	Floating-point division
FDIVR	Reverse floating-point division
FMUL	Floating-point multiplication
FSUB	Floating-point subtraction
FSUBR	Reverse floating-point subtraction

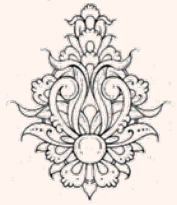
هر کدام از این دستورها به شیوه‌های متفاوتی می‌تواند ظاهر شود



دستور جمع

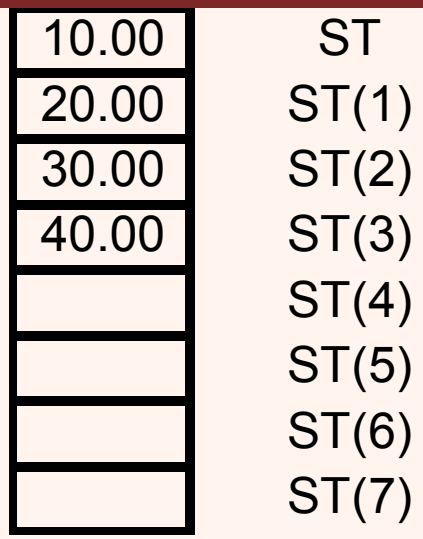
- دستورهای محاسباتی به شکل‌های مختلفی ظاهر می‌شوند، به عنوان مثال در مورد دستور جمع:
  - مقدار ثبات st را به ثبات دیگری اضافه کند.
  - مقدار ثبات دیگر و یا خانه‌ای از حافظه(که هم می‌تواند حاوی مقدار حقیقی باشد) را به st اضافه کند.

```
faddp
fadd st(num), st(0)
fadd st(0), st(num)
faddx memory(real,32 or 64)
fiadd memory (integer)
faddp st(0), st(num)
```

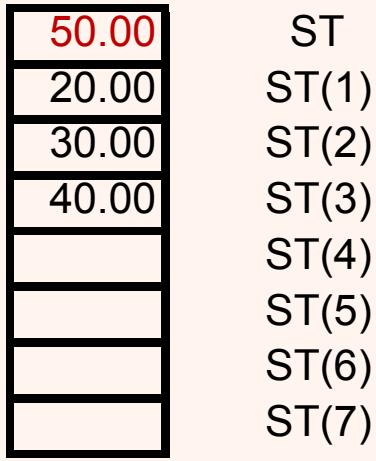


مثال

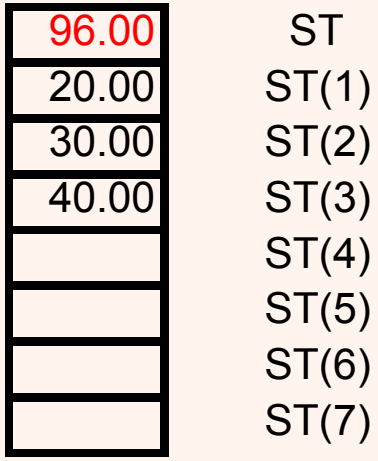
حالت اولیه



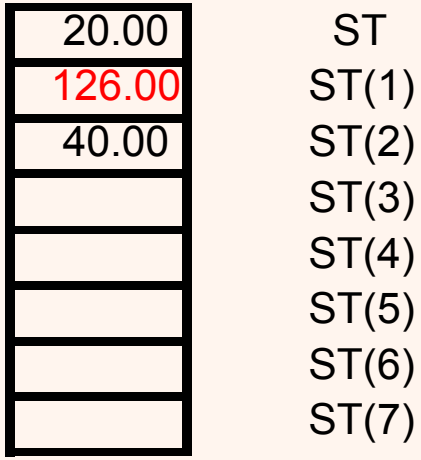

```
fadd %st(3),%st(0)
```




```
fadd fpValue
fiadd intValue
```




```
FpValue:
    .float 45.0
intValue:
    .int 1
```




```
faddp %st(0), %st(2)
```

دستور تفریق

- شکل‌های مختلف دستور تفریق بسیار شبیه به شکل‌های دستور جمع است.

```
fsubp                    st(1)=st(0)-st(1) and pop
fsub st(num), st(0)      st(0)=st(0)-st(num)
fsub st(0), st(num)      st(num)=st(0)-st(num)
fsubx memory(real, 32 or 64)
fisub memory (integer)   st(0)=st(0)-memory
fsubp st(0), st(num)     st(0)=st(0)-memory
                         st(num)=st(0)-st(num) and pop
```

نکته: نحوه‌ی به کارگیری عملوندها در فرمت AT&T در دستوراتی که خاصیت جابجایی ندارند متفاوت است

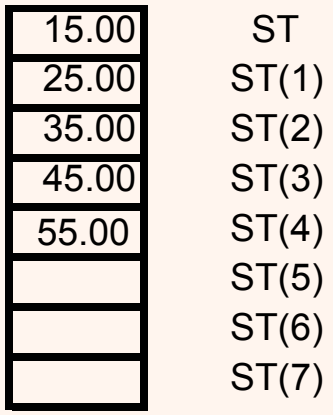
دستور تفریق (ادامه...)

- برخلاف جمع، تفریق خاصیت جابجایی ندارد.
- در تفریق، افزودن «r» به پایان دستور به معنای جابجایی ترتیب عملوندهاست.

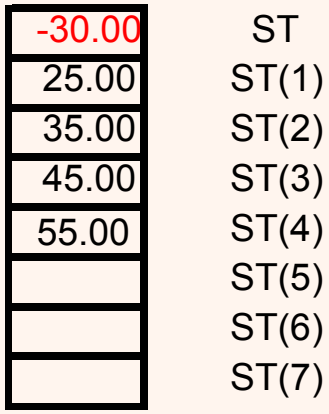
`fsubrp`	`st(1)=st(1)-st(0) and pop`
`fsubr st(num), st(0)`	`st(0)=st(num)-st(0)`
`fsubr st(0), st(num)`	`st(num)=st(num)-st(0)`
`fsubrx memory(real)`	`st(0)=memory-st(0)`
`fisubr memory (integer)`	`st(0)=memory-st(0)`
`fsubpr st(0), st(num)`	`st(num)=st(num)-st(0) and pop`

مثال

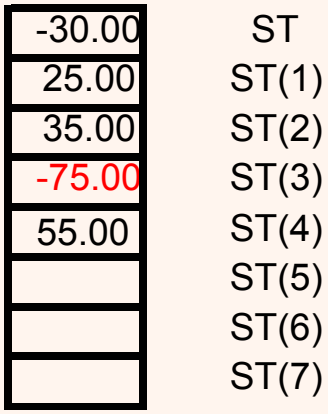
حالت اولیه



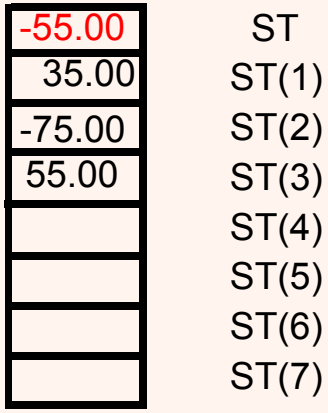
fsub %st(3), %st(0)



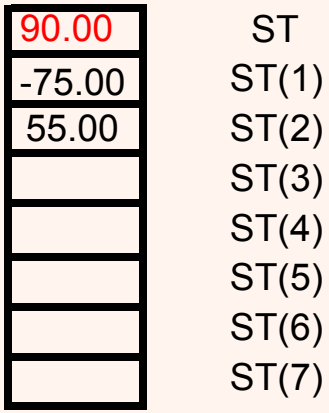
fsub %st(0), %st(3)

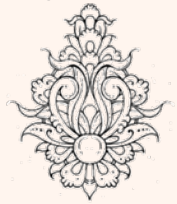


fsubp



fsubrp

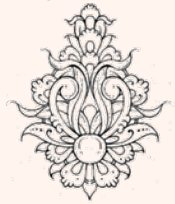




ضرب

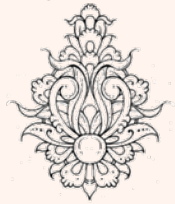
- شکل‌های مختلف دستور ضرب مانند جمع و شکل‌های مختلف دستور تقسیم مانند تفریق است.

```
fmulp
fmul st(num), st(0)
fmul st(0), st(num)
fmulx memory(real,32 or 64)
fimul memory (integer)
fmulp st(0), st(num)
```



تقسیم

```
fdivp
fdiv st(num), st(0)
fdiv st(0), st(num)
fdivx memory(real, 32 or 64)
fidiv memory (integer)
fdivp st(0), st(num)
fdivrp
fdivr st(num), st(0)
fdivr st(0), st(num)
fdivrx memory(real)
fidivr memory (integer)
fdivpr st(0), st(num)
```



مثال

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

1. Load 43.65 into ST0.
2. Divide ST0 by 22, saving the results in ST0.
3. Load 76.34 in ST0 (the answer from step 2 moves to ST1).
4. Load 3.1 in ST0 (the value in step 3 moves to ST1, and the answer from Step 2 moves to ST2).
5. Multiply ST0 and ST1, leaving the answer in ST0.
6. Add ST0 and ST2, leaving the answer in ST0 (this is the left side of the equation).
7. Load 12.43 into ST0 (the answer from Step 6 moves to ST1).
8. Multiply ST0 by 6, leaving the answer in ST0.
9. Load 140.2 into ST0 (the answer from Step 8 moves to ST1, and from Step 6 to ST2).
10. Load 94.21 into ST0 (the answer from Step 8 moves to ST2, and from Step 6 to ST3).
11. Divide ST1 by ST0, popping the stack and saving the results in ST0 (the answer from Step 8 moves to ST1, and from Step 6 to ST2).
12. Subtract ST0 from ST1, storing the result in ST0 (this is the right side of the equation).
13. Divide ST2 by ST0, storing the result in ST0 (this is the final answer).

Load 43.65 into st0

مثال

```
flds value1
```

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Divide ST0 by 22, saving the results in st0

Divide ST0 by 22, saving the results in st0

مثال

`fidiv value2`

۲

1.98409
ST(1)
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)
ST(7)

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Load 76.34 in ST0 (the answer from step 2 moves to st1)

Load 76.34 in ST0 (the answer from step 2 moves to st1)

۳

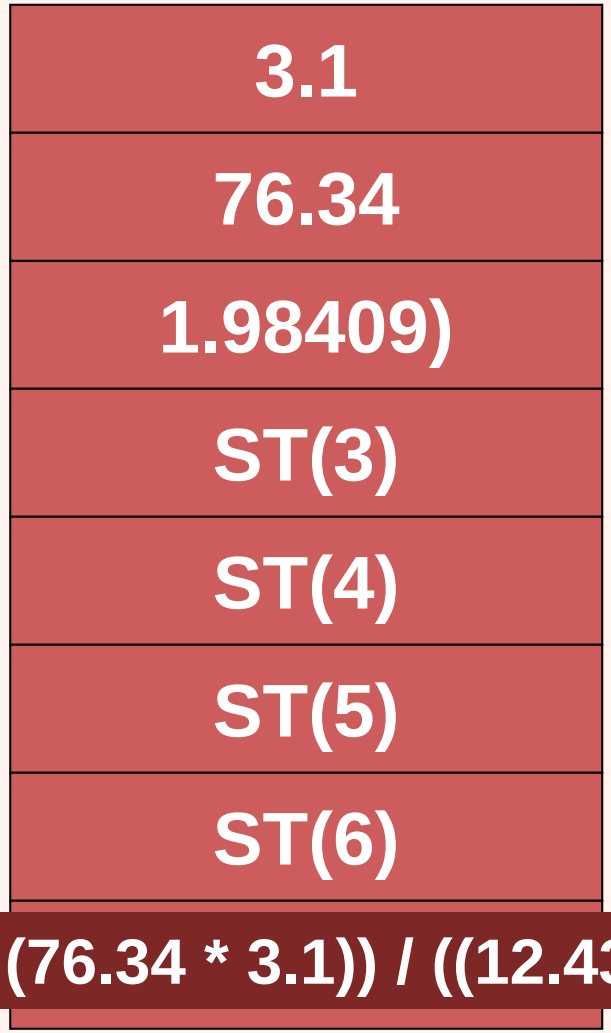
fld s value3

76.34
1.98409
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Load 3.1 in ST0 (the value in step 3 moves to ST1, and the answer from Step 2 moves to ST2).

Load 3.1 in ST0 (the value in step 3 moves to ST1, and the answer from Step 2 moves to ST2).



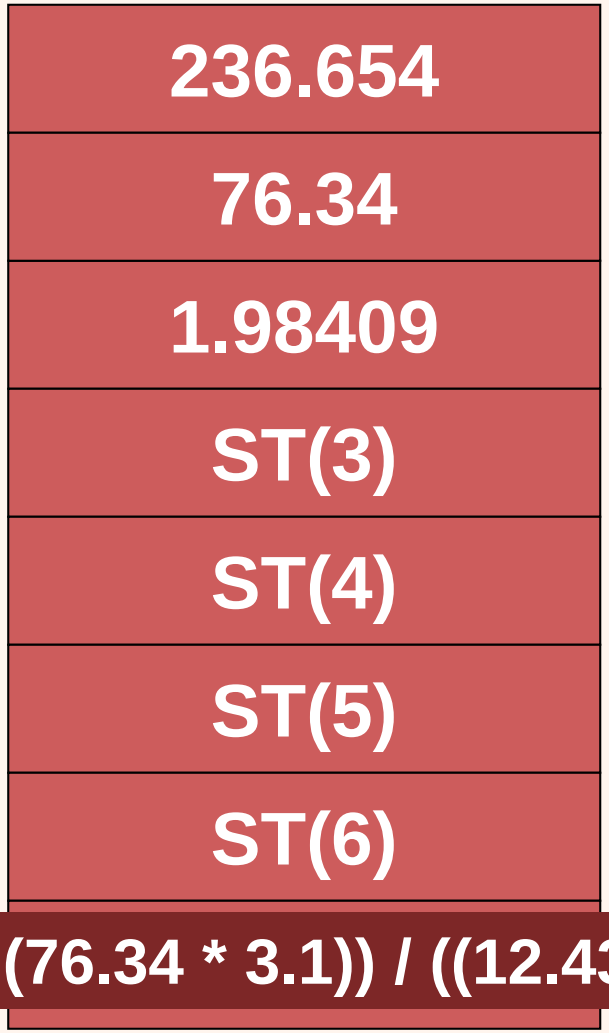
۴

```
flds value4
```

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Multiply ST0 and ST1, leaving the answer in ST0

Multiply ST0 and ST1, leaving the answer in ST0



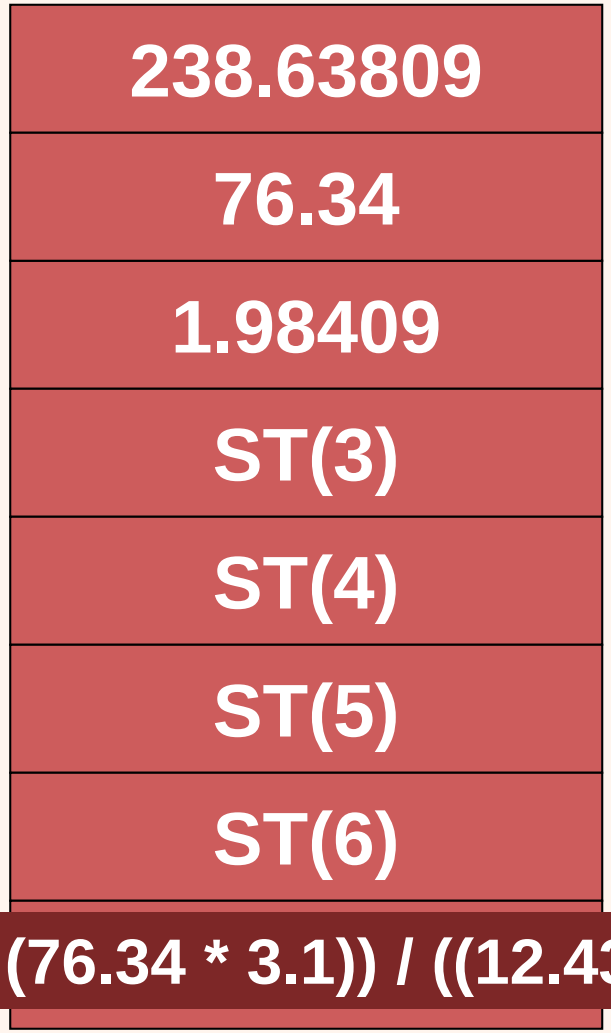

```
fmul %st(1), %st(0)
```

۵

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Add ST0 and ST2, leaving the answer in ST0 (this is the left side of the equation).

Add ST0 and ST2, leaving the answer in ST0 (this is the left side of the equation).



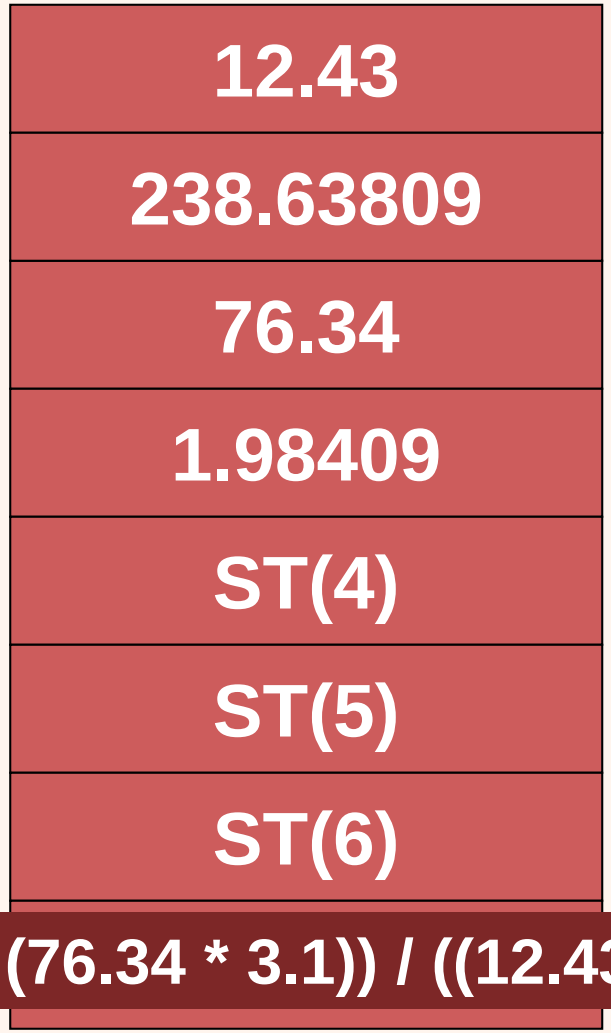
٦

```
fadd %st(2), %st(0)
```

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Load 12.43 into ST0 (the answer from Step 6 moves to ST1)

Load 12.43 into ST0 (the answer from Step 6 moves to ST1)



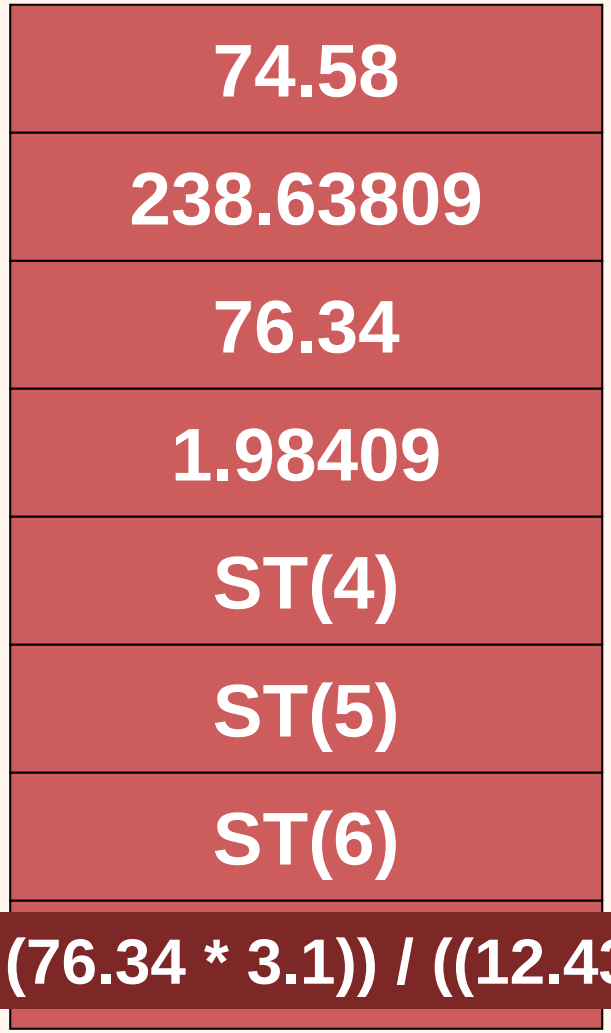
۷

```
flds value5
```

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Multiply ST0 by 6, leaving the answer in ST0

Multiply ST0 by 6, leaving the answer in ST0

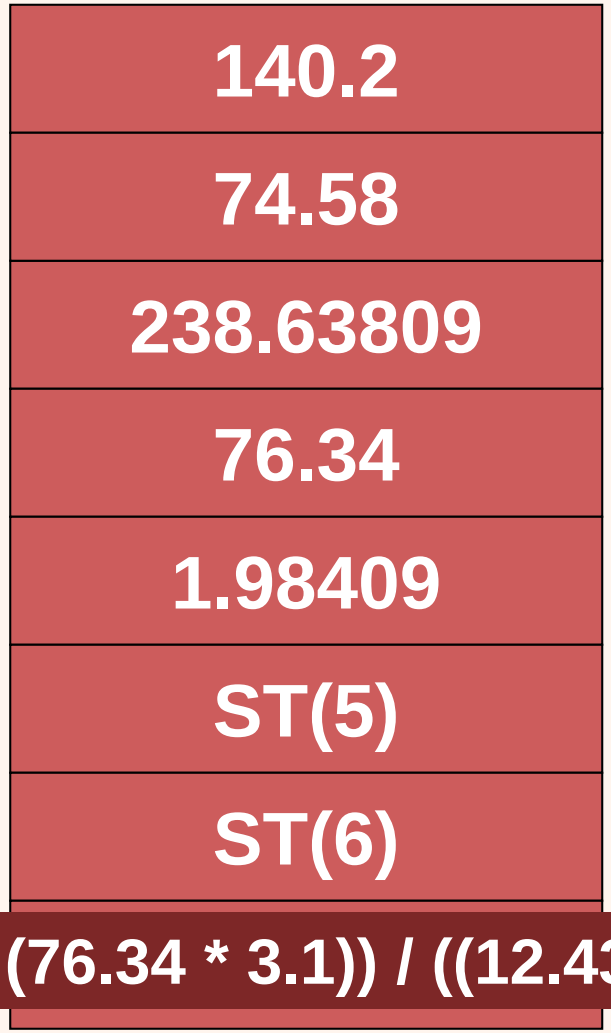



```
fimul value6
```

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Load 140.2 into ST0 (the answer from Step 8 moves to ST1, and from Step 6 to ST2)

Load 140.2 into ST0 (the answer from Step 8 moves to ST1, and from Step 6 to ST2)



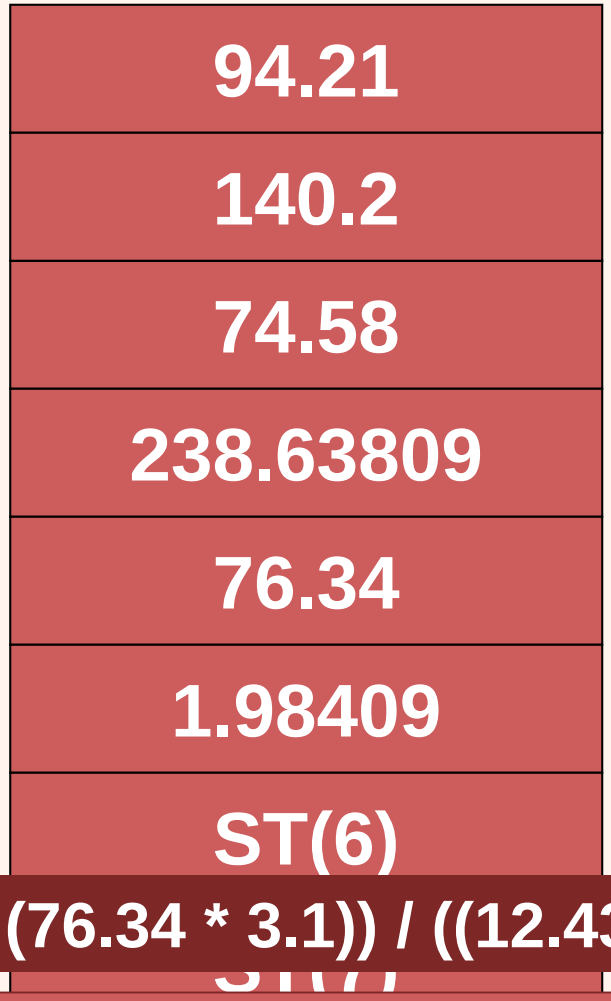
٩

`flds value7`

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Load 94.21 into ST0 (the answer from Step 8 moves to ST2, and from Step 6 to ST3)

Load 94.21 into ST0 (the answer from Step 8 moves to ST2, and from Step 6 to ST3)



10

`flds value8`

((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Divide ST1 by ST0, popping the stack and saving the results in ST0 (the answer from Step 8 moves to ST1, and from Step 6 to ST2)

Divide ST1 by ST0, popping the stack and saving the results in ST0 (the answer from Step 8 moves to ST1, and from Step 6 to ST2)

`fdivrp`

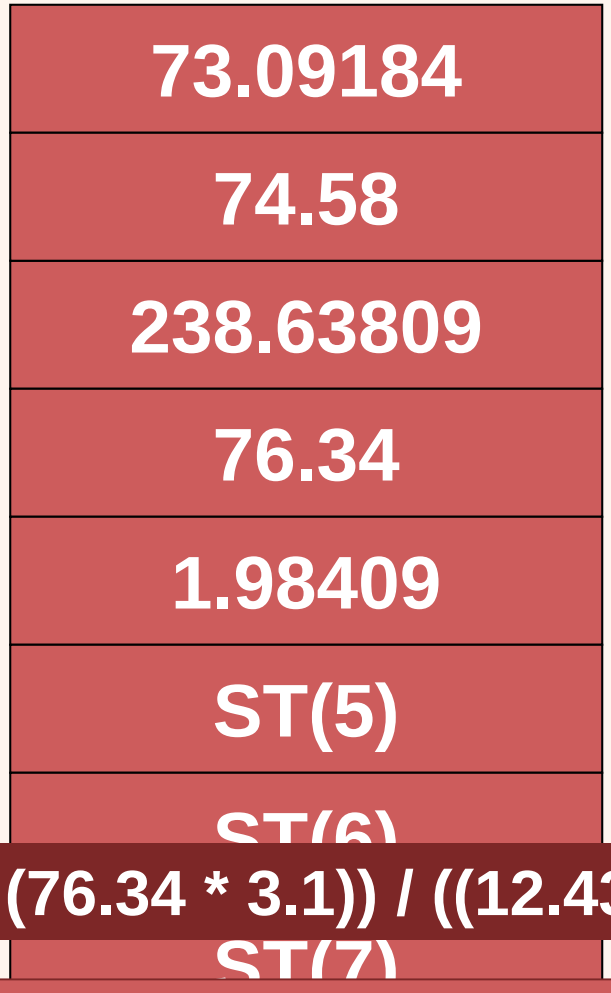
((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Subtract ST0 from ST1, storing the result in ST0 (this is the right side of the equation

Subtract ST0 from ST1, storing the result in ST0 (this is the right side of the equation

```
fsubr %st(1), %st(0)
```

۱۲



((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

Divide ST2 by ST0, storing the result in ST0 (this is the final answer).

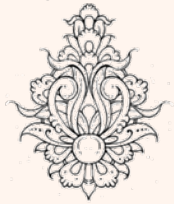
Divide ST2 by ST0, storing the result in ST0 (this is the final answer).

3.264907
74.58
238.63809
76.34
1.98409)
ST(5)
ST(6)
ST(7)

```
fdivr %st(2), %st(0)
```

۱۳

پایان



((43.65 / 22) + (76.34 * 3.1)) / ((12.43 * 6) – (140.2 / 94.21))

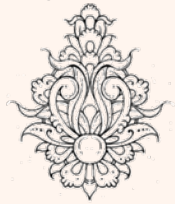
```
# fpmath1.s - An example of basic
FPU math
.section .data
value1:
   .float 43.65
value2:
   .int 22
value3:
   .float 76.34
value4:
   .float 3.1
value5:
   .float 12.43
value6:
   .int 6
value7:
   .float 140.2
value8:
   .float 94.21
output:
   .asciz "The result is %f\n"
```

```
ahmad@ubuntu:~/Courses/Assembly/ch
The result is 3.264907
```

```
.section .text
.globl _start
_start:
nop
   finit
1.    flds value1
2.    fidiv value2
3.    flds value3
4.    flds value4
5.    fmul %st(1), %st(0)
6.    fadd %st(2), %st(0)
7.    flds value5
8.    fimul value6
9.    flds value7
10.   flds value8
11.   fdivrp
12.   fsubr %st(1), %st(0)
13.   fdivr %st(2), %st(0)
   subl  $8, %esp
   fstpl (%esp)
   pushl $output
   call printf
   add  $12, %esp
   pushl $0
   call exit
```

توابع پیشرفته

- دستورهای واحد ممیز شناور محدود به دستورهای جمع، تفریق، ضرب و تقسیم نمی‌شود.
- در ادامه نگاهی گذرا به برخی از دستورهای پیشرفته خواهیم داشت:



توابع پیشرفته

Instruction	Description
F2XM1	Computes 2 to the power of the value in ST0, minus 1
FABS	Computes the absolute value of the value in ST0
FCHS	Changes the sign of the value in ST0
FCOS	Computes the cosine of the value in ST0
FPATAN	Computes the partial arctangent of the value in ST0
FPREM	Computes the partial remainders from dividing the value in ST0 by the value in ST1
FPREM1	Computes the IEEE partial remainders from dividing the value in ST0 by the value in ST1
FPTAN	Computes the partial tangent of the value in ST0
FRNDINT	Rounds the value in ST0 to the nearest integer
FSCALE	Computes ST0 to the ST1st power
FSIN	Computes the sine of the value in ST0
FSINCOS	Computes both the sine and cosine of the value in ST0
FSQRT	Computes the square root of the value in ST0
FYL2X	Computes the value ST1 * log ST0 (base 2 log)
FYL2XP1	Comp

در ادامه با برخی از این توابع آشنا خواهیم شد

زبان ماشین

مثال(fchs,fabs,fsqrt)

```
# fpmath2.s - An example of the
FABS, FCHS, and FSQRT
instructions
.section .data
value1:
    .float 395.21
value2:
    .float -9145.290
value3:
    .float 64.0
```

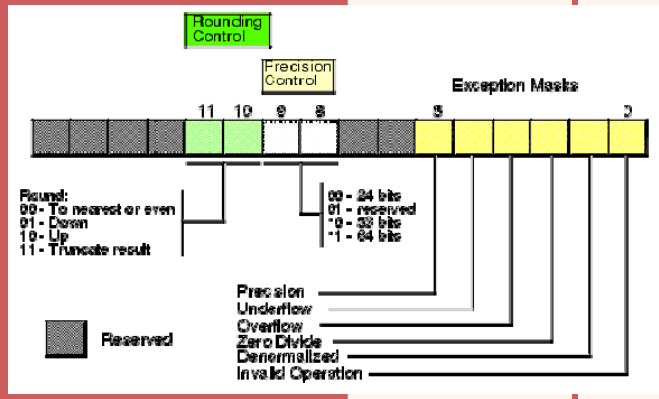
```
.section .text
.globl _start
_start:
    nop
    finit
    flds value1
    fchs
    flds value2
    fabs
    flds value3
    fsqrt

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
(gdb) info float
  R7: Valid   0xc007c59ae10000000000 -395.209991455078125
  R6: Valid   0x400c8ee5290000000000 +9145.2900390625
=>R5: Valid   0x40028000000000000000 +8
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x2800
                       TOP: 5
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x03ff
Instruction Pointer: 0x73:0x0804811a
Operand Pointer:     0x7b:0x00000000
Opcode:              0xd9fa
```

مثال(frndint)

```
.section .text
.globl _start
_start:
    nop
    finit
    flds value1
    frndint
    fists result1

    fldcw rdown
    flds value1
    frndint
    fists result2

    fldcw rup
    flds value1
    frndint
    fists result3

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
# roundtest.s - An example of
the FRNDINT instruction
.section .data
value1:
    .float 3.65
rdown:
    .byte 0x7f, 0x07
rup:
    .byte 0x7f, 0x0b
.section .bss
    .lcomm result1, 4
    .lcomm result2, 4
    .lcomm result3, 4
```




```
0x80491c4 <result1>:    0x00000004
(gdb)  x &result2
0x80491c8 <result2>:    0x00000003
(gdb)  x &result3
0x80491cc <result3>:    0x00000004
(gdb)
```

مثال(توابع مثلثاتی ۱)

```
.section .text
.globl _start
_start:
    nop
    finit
    flds degree1
    fidivs val180
    fldpi
    fmul %st(1), %st(0)
    fsts radian1
    fsin
    fsts result1
    flds radian1
    fcos
    fsts result2

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
# trigtest1.s - An example of
using the FSIN and FCOS
instructions
.section .data
degree1:
    .float 90.0
val180:
    .int 180
.section .bss
    .lcomm radian1, 4
    .lcomm result1, 4
    .lcomm result2, 4
```

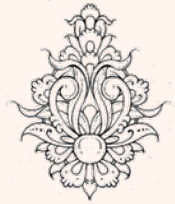
```
0x80491c0 <radian1>:    1.57079637
(gdb) x /f &result1
0x80491c4 <result1>:    1
(gdb) x /f &result2
0x80491c8 <result2>:    -4.37113883e-08
(gdb)
```

مثال(توابع مثلثاتی ۲)

```
.globl _start
_start:
    nop
    finit
    flds degree1
    fidivs val180
    fldpi
    fmul %st(1), %st(0)
    fsincos
    fstps cosresult
    fsts sinresult

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

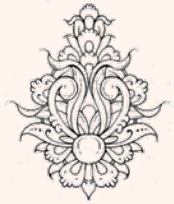
```
# trigtest2.s - An example of
using the FSINCOS instruction
.section .data
degree1:
    .float 90.0
val180:
    .int 180
.section .bss
    .lcomm sinresult, 4
    .lcomm cosresult, 4
.section .text
```

```
(gdb) x /f &sinresult
0x80491ac <sinresult>:  1
(gdb) x /f &cosresult
0x80491b0 <cosresult>:  -2.71050543e-20
```



پرش شرطی در واحد ممیز شناور

- در واحد ممیز شناور، دستوری برای مقایسه وجود دارد.
- در تمامی این دستورها محتوای st با عملوند دیگری مقایسه می‌شود.



پرش شرطی در برنامه‌نویسی

`fcom`	compares st and st(1)
`fcom st(num)`	compares st and st(num)
`fcom memory(real)`	compares st and a real in memory
`ficom memory(int)`	compares st and an int in memory
`ftst (none)`	compares st and 0.0
`fcomp (none)`	compares st and st(1) pops stack.
`fcomp st(num)`	compares st and st(num) pops stack.
`fcomp memory(real)`	compares st and a real in memory pops stack.
`ficomp memory(int)`	compares st and an int in memory pops stack.
`fcompp (none)`	compares st and st(1) pops stack. pops stack.

پرش شرطی در واحد ممیز شناور

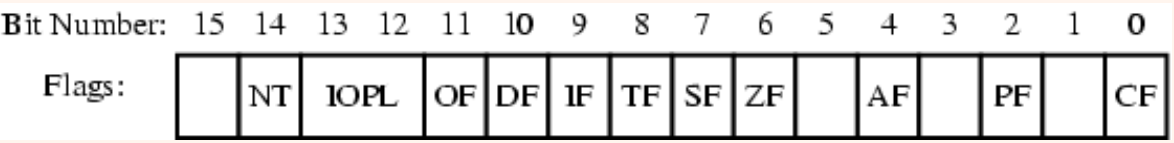
Result of comparison	C3	C2	C0
ST > second operand	0	0	0
ST < second operand	0	0	1
ST = second operand	1	0	0

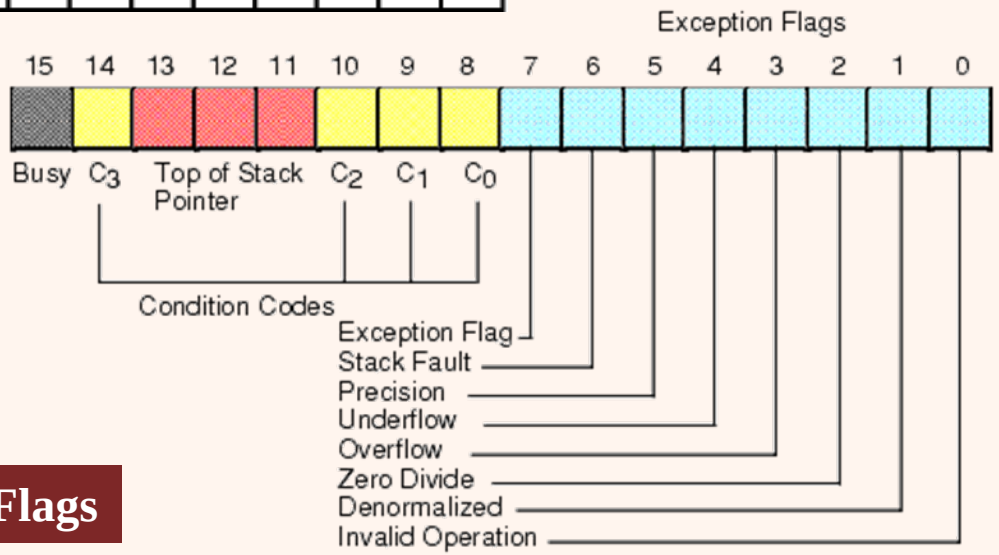
در صورتی که عملگرها قابل مقایسه نباشند (نامرتب) ، هر سه بیت ۱ خواهند شد.

- برای استفاده از نتیجه‌ی مقایسه باید ثبات وضعیت به ثبات پرچم منتقل شود.

`fstsw` **Store Floating-Point Status Word**

پرش شرطی در واحد ممیز شناور (ادامه...)





sahf

Store AH into Flags

این دستور بعد از `fstsw %ax` مورد استفاده قرار می‌گیرد.
بیت 0، 2، 4، 6 و 7 از ثبات ah به ثبات پرچم‌ها منتقل می‌شود

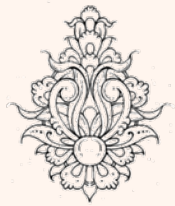
C0	Carry Flag
C2	Parity Flag
C3	Zero Flag

مثال

```
# fcomtest.s - An example of the FCOM instruction
.section .data
value1:
    .float 10.923
value2:
    .float 4.5532
.section .text
```

```
_start:
    nop
    flds valu
    fcoms value2
    fstsw
    sahf
    ja greater
    jb lessthan
    movl $1, %eax
    movl $0, %ebx
    int $0x80
greater:
    movl $1, %eax
    movl $2, %ebx
    int $0x80
lessthan:
    movl $1, %eax
    movl $1, %ebx
    int $0x80
```

```
ahmad@ubuntu:~/Courses/Assembly/chapter6$ echo $?
2
```



هشدار: بررسی تساوی دو عدد ممیز شناور اصلا ایده‌ی خوبی نیست

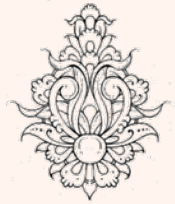
پرش شرطی در واحد ممیز شناور (ادامه...)

- با پیدایش Pentium Pro، دستورهای مقایسه‌ی اعشاری و انتقال به ثبات‌ها در هم ادغام شدند.

```
fcomi
```
compares st and st(1)

```
fcomip st(num)
```
compares st and st(num) pops stack.

Result of comparison	ZF	PF	CF
ST > second operand	0	0	0
ST < second operand	0	0	1
ST = second operand	1	0	0

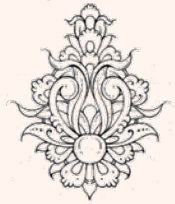


مثال

```
# fcomitest.s - An example of
the FCOMI instruction
.section .data
value1:
   .float 10.923
value2:
   .float 4.5532
.section .text
.globl _start
```

```
_start:
   nop
   flds value2
   flds value1
   fcomi %st(1), %st(0)
   ja greater
   jb lessthan
   movl $1, %eax
   movl $0, %ebx
   int $0x80
greater:
   movl $1, %eax
   movl $2, %ebx
   int $0x80
lessthan:
   movl $1, %eax
   movl $1, %ebx
   int $0x80
```

```
ahmad@ubuntu:~/Courses/Assembly/chapter6$ ./fcomites
ahmad@ubuntu:~/Courses/Assembly/chapter6$ echo $?
2
```



دستورهای جابجایی شرطی

مانند جابجایی شرطی برای اعداد صحیح، در اینجا نیز جابجایی‌ها بر اساس ثبات پرچم‌ها صورت می‌پذیرد. از این رو این دستورها پس از FCOMI به کار می‌روند

Instruction	Description
FCMOVB	Move if ST(0) is below ST(x).
FCMOVE	Move if ST(0) is equal to ST(x).
FCMOVBE	Move if ST(0) is below or equal to ST(x).
FCMOVU	Move if ST(0) is unordered.
FCMOVNB	Move if ST(0) is not below ST(x).
FCMOVNE	Move it ST(0) is not equal to ST(x).
FCMOVNBE	Move if ST(0) is not below or equal to ST(x).
FCMOVNU	Move if ST(0) is not unordered.

fcmovxx st(num), st(0)

مثال

```
# fcmovtest.s - An example
of the FCMOVxx instructions
.section .data
value1:
    .float 20.5
value2:
    .float 10.90
.section .text
.globl _start
_start:
    nop
    finit
    flds value1
    flds value2
    fcomi %st(1), %st(0)
    fcmovb %st(1), %st(0)

    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
  R7: Valid   0x4003a400000000000000 +20.5
=>R6: Valid   0x4002ae66660000000000 +10.89999961853027344
  R5: Empty   0x00000000000000000000
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x3000
                       TOP: 6
```

```
  R7: Valid   0x4003a400000000000000 +20.5
=>R6: Valid   0x4003a400000000000000 +20.5
  R5: Empty   0x00000000000000000000
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x3000
                       TOP: 6
```

نکاتی در مورد محاسبات ممیز شناور

- یکی از زمان‌برترین بخش‌های برنامه‌ی شما مربوط به این واحد می‌شود، برای بهبود کارایی این قسمت:
  - مراقب باشید سرریز با فروریز رخ ندهد.
  - استفاده از محاسبات با دقت معمولی (single) توصیه می‌شود.
  - می‌توان از جدول مراجعه (look up table) استفاده کرد.
  - از دستورهای جدید بهره جست.
  - هنگامی که محاسبات بر روی ترکیبی از اعداد صحیح و اعشاری انجام می‌شود، بارگذاری عدد صحیح و انجام عملیات سریع‌تر از دستورهای محاسباتی بر روی اعداد صحیح خواهد بود.