КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 15 Базовый FPU. Команды.

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2022

2022.04.27

Оглавление

Команды FPU

Код операции всех команд FPU начинается с 5 бит: 11011.
Мнемоника команд FPU начинается с буквы «F»: **FIST**, **FADD**

В инструкциях с плавающей точкой ассемблер **nasm** позволяет использовать либо форму с одним операндом, либо с двумя. При этом для ясности предоставляется ключевое слово **TO**. В результате можно писать или

```
fadd   st0, st1  ; ST(0)  <-- ST(0) + ST(1)
fadd   st1, st0  ; ST(1)  <-- ST(1) + ST(0)
```

или можно использовать альтернативную форму с одним операндом

```
fadd   st1       ; ST(0)  <-- ST(0) + ST(1)
fadd   to st1    ; ST(1)  <-- ST(1) + ST(0)
```

Команды загрузки констант

FLD1 — поместить в стек 1.0; **FLDZ** — поместить в стек +0.0; **FLDPI** — поместить в стек число π; **FLDL2E** — поместить в стек $\log_2(e)$; **FLDL2T** — поместить в стек $\log_2(10)$; **FLDLN2** — поместить в стек $\ln 2$; **FLDLG2** — поместить в стек $\lg 2$.	Команды помещают в стек соответствующую константу. Начиная с i80387 все константы хранятся в более точном формате, нежели 80-битное вещественное число и при загрузке в стек происходит округление в соответствии с полем RC.

После последовательного выполнения вышеприведенных инструкций имеем (окно edb):

```
       R0 ST7 empty     0.0
TOP--> R1 ST0 valid     0.30102999566398119523
       R2 ST1 valid     0.69314718055994530943
       R3 ST2 valid     3.3219280948873623478
       R4 ST3 valid     1.4426950408889634074
       R5 ST4 valid     3.1415926535897932385
       R6 ST5 zero      0.0
       R7 ST6 valid     1.0

FTR   1003      3 2 1 0       E S P U O Z D I
FSR   0800 Cond 0 0 0 0   Err 0 0 0 0 0 0 0 0
FCR   037F Prec NEAR 64   Mask    1 1 1 1 1 1
```

Команды пересылки данных

FLD src — загрузить вещественное число в стек.

```
section  .data
opdd     dd 1.1234567e20 ;
opdq     dd 1.1234567e20 ;
opdt     dt 1.1234567e20 ;

section  .code
fld      tword [opdt] ; 1.12334567e+20
fld      qword [opdq] ; 1.12334567000000041e+20
fld      dword [opdd] ; 1.12334597350835241e+20
```

Помещает содержимое источника в стек и декрементирует TOP.

В качестве источника используется 32-, 64-, 80-битная переменная или ST(n).

Команда **`FLD ST(0)`** делает копию вершины стека.

FST dst — скопировать вещественное число из стека (вершина стека **TOP** не меняется).
FSTP dst — считать вещественное число из стека (число выталкивается из стека — **TOP**++).

Копирует **ST(0)** в приемник
FST — приемник 32, 64 бит или пустой **ST(n)**
FSTP — приемник 32, 64, 80 бит или пустой **ST(n)**.
FSTP — помечает **ST(0)** как пустой и инкрементирует **TOP**.

FILD src — загрузить целое число в стек.

Преобразовывает целое со знаком из источника (`int16_t`, `int32_t`, `int64_t`) в вещественный формат, помещает результат на вершину стека и декрементирует **TOP**.

FIST dst — скопировать целое число из стека.
FISTP dst — считать целое число из стека (w/pop).

Преобразовывает число с вершины стека в целое со знаком и записывает его в приемник.
FIST — приемник 16, 32 бит;
FISTP — приемник 16, 32, 64 бит;
FISTP помечает **ST(0)** как пустой и инкрементирует **TOP**.
При попытке записи слишком большого числа, бесконечности или **NAN**, приводит к исключению «недопустимая операция».
Если **IM = 1**, будет записана целая неопределенность.

FBLD src — загрузить десятичное число в стек.

Преобразовывает BCD-число из источника (80-бит переменная в памяти) и помещает на вершину стека и декрементируетTOP.

FBSTP dst — считать десятичное число из стека.

Преобразовывает число с вершины стека в 80-битное упакованное десятичное и записывает его в приемник (80-бит переменная) и выталкивает число из стека (помечает **ST(0)** как пустой и инкрементирует TOP.)

Попытка записи слишком большого числа, бесконечности или NAN, приводит к исключению «недопустимая операция» и записи десятичной неопределенности, если **IM=1**.

FXCH src — обменять местами два регистра стека.

Обмен содержимого вершины стека **ST(0)** и регистра **ST(n)**. Если операнд не указан, выполняется обмен **ST(0)** и **ST(1)**.

FCMOVcc dst, src — условная пересылка данных (P6).

Содержимое источника копируется в приемник, если выполняется необходимое условие.

В качестве источника используются только регистры **ST(n)**.

В качестве приемника используется только регистр **ST(0)**.

Команда работает по значениям флагов из регистра FLAGS ЦПУ, поэтому обработать результат команды сравнения операндов в сопроцессоре (например, **FCOM**), необходима следующая последовательность команд:

```
fcom  (или другая команда сравнения) ; установить C0, C2, C3 по рез-там сравнения
fstsw ax                              ; сохранить C0, C2, C3 в AX
sahf                                  ; загрузить AH в EFLAGS
```

При этом флаги **C0**, **C2**, **C3** будут загружены в регистр флагов ЦПУ и последующая команда **FCMOVcc** обработает результат предыдущего сравнения в FPU.

Команда	**Значения флагов**	**действие после FCOM**
FCMOVE	ZF = 1	если равно (==)
FCMOVNE	ZF = 0	если не равно (!=)
FCMOVB	CF = 1	если меньше (<)
FCMOVBE	CF = 1 и XF = 1	если меньше или равно (<=)
FCMOVNB	CF = 0	если не меньше (>=)
FCMOVNBE	CF = 0 и ZF = 0	если не меньше или равно (>)
FCMOVU	PF = 1	если не сравнимы
FCMOVNU	PF = 0	если сравнимы

Базовая арифметика

FADD dst, src — сложение вещественных чисел;
FADDP dst, src — сложение вещественных чисел с выталкиванием из стека;
FIADD src — сложение целых чисел;

Выполняется сложение источника и приемника, после чего результат помещается в приемник. **FADDP** дополнительно выталкивает **ST(0)** из стека (помечает **ST(0)** как пустой и инкрементирует **TOP**).

Формы команд сложения

FADD src — источником является 32-бит 64-бит переменная, содержащая вещественное число, приемник — **ST(0)**.

FADD ST(0), ST(n) — операнды в стеке FPU заданы явно;
FADD ST(n), ST(0) — операнды в стеке FPU заданы явно;
FADDP ST(n), ST(0) — операнды в стеке FPU заданы явно;
FADD без операндов эквивалентна **FADD ST(0), ST(1)**
FADDP без операндов эквивалентна **FADDP ST(1), ST(0)**
FIADD src — источником является 16-бит 32-бит переменная, содержащая целое число, приемник — **ST(0)**.

FSUB dst, src — вычитание вещественных чисел;
FSUBP dst, src — вычитание вещественных чисел с выталкиванием из стека;
FISUB src — вычитание целых чисел;

Выполняется вычитание источника из приемника, после чего результат помещается в приемник. **FSUBP** дополнительно выталкивает **ST(0)** из стека (помечает **ST(0)** как пустой и инкрементирует **TOP**).

Формы команд вычитания

FSUB src — источником является 32-бит 64-бит переменная, содержащая вещественное число, приемник — **ST(0)**.

FSUB ST(0), ST(n) — операнды в стеке FPU заданы явно;
FSUB ST(n), ST(0) — операнды в стеке FPU заданы явно;
FSUBP ST(n), ST(0) — операнды в стеке FPU заданы явно;
FSUB без операндов эквивалентна **FSUB ST(0), ST(1)**
FSUBP без операндов эквивалентна **FSUBP ST(1), ST(0)**
FISUB src — источником является 16-бит 32-бит переменная, содержащая целое число, приемник — **ST(0)**.

Если один из операндов — **INF**, результат будет **INF** соответсвующего знака.

Если оба операнда — **INF** одного знака, результат неопределен (исключение «недопустимая операция»).

FSUBR dst, src — обратное вычитание вещественных чисел;
FSUBRP dst, src — обратное вычитание с выталкиванием из стека;
FISUBR src — обратное вычитание целых чисел;

Эквивалентны **FSUB/FSUBP/FISUB**, но выполняют вычитание приемника из источника, а не источника из приемника.

FMUL dst, src — умножение вещественных чисел;
FMULP dst, src — умножение вещественных чисел с выталкиванием из стека;
FIMUL src — умножение целых чисел;

Выполняется умножение источника и приемника, после чего результат помещает в приемник. **FMULP** дополнительно выталкивает **ST(0)** из стека (помечает **ST(0)** как пустой и инкрементирует **TOP**).

Формы команд умножения

FMUL src — источником является 32-бит 64-бит переменная, содержащая вещественное число, приемник — **ST(0)**.

FMUL ST(0), ST(n) — операнды в стеке FPU заданы явно;
FMUL ST(n), ST(0) — операнды в стеке FPU заданы явно;
FMULP ST(n), ST(0) — операнды в стеке FPU заданы явно;
FMUL без операндов эквивалентна **FMUL ST(0), ST(1)**
FMULP без операндов эквивалентна **FMULP ST(1), ST(0)**
FIMUL src — источником является 16-бит 32-бит целая переменная, содержащая целое число, приемник — **ST(0)**.

FDIV dst, src — деление вещественных чисел;
FDIVP dst, src — деление вещественных чисел с выталкиванием из стека;
FIDIV src — деление целых чисел;

Выполняется деление приемника на источник, после чего результат помещается в приемник. **FDIVP** дополнительно выталкивает **ST(0)** из стека (помечает **ST(0)** как пустой и инкрементирует **TOP**).

Формы команд деления

FDIV src — источником является 32-бит 64-бит переменная, содержащая вещественное число, приемник — **ST(0)**.

FDIV ST(0), ST(n) — операнды в стеке FPU заданы явно;
FDIV ST(n), ST(0) — операнды в стеке FPU заданы явно;
FDIVP ST(n), ST(0) — операнды в стеке FPU заданы явно;
FDIV без операндов эквивалентна **FDIV ST(0), ST(1)**
FDIVP без операндов эквивалентна **FDIVP ST(1), ST(0)**
FIDIV src — источником является 16-бит 32-бит переменная, содержащая целое число, приемник — **ST(0)**.

При делении бесконечности на ноль или на любое другое число результат — бесконечность.
При делении нуля на бесконечность или на любое другое число результат — ноль.
При делении на ноль нормального числа происходит исключение деления на ноль, а если флаг **ZM = 1**, результат — бесконечность соответствующего знака.

FDIVR dst, src – обратное деление вещественных чисел;
FDIVRP dst, src – обратное деление с выталкиванием из стека;
FIDIVR src – обратное деление целых чисел;

Эквивалентны **FDIV**/**FDIVP**/**FIDIV**, но выполняют деление источника на приемник, а не приемника на источник.

FPREM — найти частичный остаток от деления
FPREM1 — найти частичный остаток от деления в стандарте IEEE

Эти команды выполняют деление **ST(0)** на **ST(1)** и помещают остаток от деления в **ST(0)**.

Деление выполняется как несколько последовательных вычитаний, но не более 64 за одну команду.

```
Remainder := ST(0) – (Q * ST(1))
```

Инструкция **FPREM** получила название «частичный остаток» из-за способа вычисления остатка — эта инструкция получает остаток посредством *итеративного вычитания*. Однако он может уменьшить показатель **ST(0)** не более чем на 63 за одно выполнение инструкции.

Если **ST(0)** не стал меньше **ST(1)**, говорят, что получен частичный остаток. Показатель частичного остатка будет меньше, чем показатель исходного делимого, по крайней мере, на 32.

Если был получен точный остаток, флаг **C2** сбрасывается, если частичный — устанавливается.

Программное обеспечение может повторно выполнить инструкцию (используя частичный остаток в **ST(0)** в качестве делимого), пока **C2** не будет сброшен.

Следует обратить внимание, что при выполнении такого цикла вычисления остатка процедура прерывания с более высоким приоритетом, которой требуется FPU, может вызвать переключение контекста между инструкциями в цикле.

Важное применение инструкции **FPREM** — уменьшение аргументов периодических функций. Когда сокращение завершено, инструкция сохраняет три младших бита частного во флагах **C0**, **C3** и **C1** слова состояния FPU. Эта информация важна для уменьшения аргумента для функции тангенса (с использованием модуля **π/4**), поскольку она определяет местонахождение исходного угла в правильном одном из восьми секторов единичной окружности.

Различие между **FPREM** и **FPREM1** состоит в том, как округляется частное.

FPREM и **FPREM1** выполняют деление **ST(0)** на **ST(1)**, затем выполняют округление — **FPREM** к нулю, **FPREM1** — к ближайшему целому.

Если в результате частное **Q** меньше 64, вычисляют точный остаток, если больше — частичный.

Работа этих инструкции одинакова в не 64-битном и 64-битном режимах.

FABS — найти абсолютное значение.
Если **ST(0) < 0**, переводит его в положительное.

FCHS — изменить знак.
Меняет знак числа в **ST(0)**.

FRNDINT — округлить до целого.
Округляет **ST(0)** в соответствии в режимом округления **RC** в регистре **CR**.

FXTRACT — извлечь экспоненту и мантиссу.
Разделяет число в **ST(0)** на мантиссу и экспоненту, сохраняет экспоненту **ST(1)**, а мантиссу в **ST(0)**. Сначала экспонента помещается в **ST(0)**, потом декрементируется TOP и в **ST(0)** помещается мантисса.

Эта инструкция и инструкция **F2XM1**[1] полезны для выполнения операций возведения в степень и масштабирования диапазона. Инструкция **FXTRACT** также полезна для преобразования чисел в формате с плавающей запятой двойной расширенной точности в десятичные представления (например, для печати или отображения).

Если замаскировано исключение деления нуля с плавающей запятой (**#Z**), а исходный операнд равен нулю, значение экспоненты, равное бесконечности, сохраняется в регистре **ST(1)**, а в регистре **ST(0)** сохраняется 0 со знаком исходного операнда.

1 2^x-1

FSCALE — масштабировать по степеням двойки.

Усекает значение в исходном операнде **ST(1)** в сторону 0 до целого значения и добавляет это значение к экспоненте целевого операнда **ST(0)**.

Эта инструкция обеспечивает быстрое умножение или деление на целые степени числа 2.

В большинстве случаев изменяется только показатель степени, а мантисса (мантисса) остается неизменной. Однако, когда значение, масштабируемое в **ST(0)**, является денормализованным, мантисса тоже изменяется, и результат может оказаться нормализованным числом.

Точно так же, если в результате операции масштабирования возникает переполнение или потеря значимости, результирующая мантисса будет отличаться от мантиссы источника.

Инструкцию FSCALE также можно использовать для действия, обратного инструкции **FXTRACT**, как показано в следующем примере:

```
    FXTRACT    ; ST(0) <- мантисса, ST(1) <- порядок
    FSCALE     ;
    FSTP ST(1) ; ST(0) -> ST(1) & POP ST(0)
```

В этом примере инструкция **FXTRACT** извлекает мантиссу и показатель степени из значения в **ST(0)** и сохраняет их в **ST(0)** и **ST(1)** соответственно. Затем **FSCALE** масштабирует мантиссу в **ST(0)** по экспоненте в **ST(1)**, воссоздавая то значение, которое было перед выполнением операции **FXTRACT**. Инструкция **FSTP ST(1)** перезаписывает показатель степени (извлеченный инструкцией FXTRACT) воссозданным значением, которое возвращает стек в исходное состояние с занятым только одним регистром **ST(0)**.

FSQRT — извлечь квадратный корень

Извлекает квадратный корень из **ST(0)** и сохраняет результат в **ST(0)**.

Команды сравнения FPU

FCOM src — сравнить вещественные числа;

FCOMP src — сравнить вещественные числа и вытолкнуть;

FCOMPP — сравнить и вытолкнуть два числа;

Команды выполняют сравнение содержимого **ST(0)** с источником (32-бит или 64-бит переменная в памяти, или **ST(n)**), если операнд не указан — **ST(1)**, и устанавливают флаги **C0**, **C2**, **C3**.

Условие	C3	C2	C0
ST(0) > src	0	0	0
ST(0) < src	0	0	1
ST(0) = src	1	0	0
Несравнимы	1	1	1

Если один из операндов **NAN** или неподдерживаемое число, возникает исколючение «недопустимая операция» (**#IA**).

Если оно замаскировано (**IM = 1**), все три флага устанавливаются в 1 (несравнимость).

После команд сравнения используя **FSTSW** и **SAHF** можно перевести флаги **C3**, **C2**, **C0** в **ZF**, **ZP** и **CF**, соответственно, после чего все условные команды могут использовать результат сравнения, как после команды **CMP**.

FCOMP выталкивает из стека содержимое **ST(0)** и помечает его как пустой.

FCOMPP выталкивает из стека оба своих операнда **ST(0)** и **ST(1)**.

FUCOM **src** — сравнить без учета упорядочения;
FUCOMP **src** — сравнить без учета упорядочения с выталкиванием из стека;
FUCOMPP — сравнить без учета упорядочения с выталкиванием двух чисел;

При данном сравнении проверяется класс сравниваемых **NaN**.

Команды аналогичны командам **FCOM/FCOMP/FCOMPP**, но в роли источников могут выступать только регистры **ST(n)**.

Условие	C3	C2	C0
ST(0) > ST(i)	0	0	0
ST(0) < ST(i)	0	0	1
ST(0) = ST(i)	1	0	0
Несравнимы	1	1	1

Единственное отличие состоит в том, что инструкции **FUCOM**/**FUCOMP**/**FUCOMPP** вызывают исключение недопустимого арифметического операнда (**#IA**) только тогда, когда один или оба операнда являются **SNaN** или имеют неподдерживаемый формат;

Если один из операндов **QNAN** (тихое не-число), флаги **C3**, **C2**, **C0** устанавливаются в 1, но исключение «недопустимая операция» не происходит.

Команды **FCOM**/**FCOMP**/**FCOMPP** вызывают исключение недопустимой операции, когда один или оба операнда имеют значение **NaN** любого типа или находятся в неподдерживаемом формате.

FCOMI **src** — сравнить и установить **EFLAGS**;
FCOMIP **src** — сравнить, установить **EFLAGS** и вытолкнуть;
FUCOMI **src** — сравнить без учета упорядочения и установить **EFLAGS** из стека;
FUCOMIP **src** — сравнить без учета упорядочения, установить **EFLAGS** и вытолкнуть из стека.

Команды выполняют сравнение содержимого **ST(0)** с источником **ST(n)** и устанавливают флаги регистра **EFLAGS**.

Условие	ZF	PF	CF
ST(0) > src	0	0	0
ST(0) < src	0	0	1
ST(0) = src	1	0	0
Несравнимы	1	1	1

Команды эквивалентны **FCOM/FCOMP/FUCOM/FUCOMP**, вслед за которыми выполняются команды **FSTSW AX** и **SAHF**, но они не изменяют содержимое **AX** и выполняются быстрее.

FICOM **src** — сравнить целые числа;
FICOMP **src** — сравнить целые и вытолкнуть из стека;

Команды сравнивают содержимое **ST(0)** и источника, в качестве которого используются ячейка памяти, содержащая целое число `int16_t` или `int32_t`.

В остальном команды эквивалентны **FCOM**/**FCOMP**.

FTST — Проверить, не содержит ли **ST(0)** нуль.

Сравнивает **ST(0)** с нулем и выставляет флаги **C3**, **C2**, **C0** аналогично другим командам сравнения.

FXAM — проанализировать содержимое **ST(0)**.

Устанавливает флаги **C3**, **C2**, **C0** в зависимости от типа числа, находящегося в **ST(0)**, в соответствии с правилами, приведенными в таблице.

Тип числа	C3	C2	C0
Неподдерживаемое	0	0	0
Не-число	0	0	1
Нормальное конечное число	0	1	0
бесконечность	0	1	1
Ноль	1	0	0
Регистр пуст	1	0	1
Денормализованное число	1	1	0

Трансцендентные операции FPU

FSIN — синус.

Вычисляет синус числа в **ST(0)** и сохраняет результат там же.

Операнд считается заданным в радианах и не может быть > 2^{63} или < 2^{-63}.

Если операнд велик, можно использовать **FPREM** с делителем 2π для приведения.

Если операнд выходит за указанные пределы, устанавливается флаг **C2**, а содержимое **ST(0)** не изменяется.

FCOS — косинус.

Вычисляет косинус числа в **ST(0)** и сохраняет результат там же.

Операнд считается заданным в радианах и не может быть > 2^{63} или < 2^{-63}.

Если операнд велик, можно использовать **FPREM** с делителем 2π для приведения.

Если операнд выходит за указанные пределы, устанавливается флаг **C2**, а содержимое **ST(0)** не изменяется.

FSINCOS — синус и косинус.

Вычисляет синус и косинус числа в **ST(0)**, помещает синус в **ST(1)**, а косинус — в **ST(0)**. Операнд считается заданным в радианах и не может быть > 2^{63} или < 2^{-63}.

Если операнд выходит за указанные пределы, устанавливается флаг **C2**, а содержимое **ST(0)** не изменяется.

FPTAN — Тангенс.

Вычисляет тангенс числа, находящегося в **ST(0)**, заменяет его на вычисленное значение и помещает в стек 1, так что по окончании команды результат оказывается в **ST(1)**, а **ST(0)** содержит 1.

Операнд считается заданным в радианах и не может быть > 2^{63} или < 2^{-63}.

Если операнд выходит за указанные пределы, устанавливается флаг **C2**, а содержимое **ST(0)** не изменяется.

Единица помещается в стек для того, чтобы при необходимости можно было командой **FDIVR** получить котангенс.

FPATAN — Частичный арктангенс

Вычисляет арктангенс числа, получаемого при делении **ST(1)** на **ST(0)**, и сохраняет результат в **ST(1)** и выталкивает **ST(0)**.

Результат всегда имеет знак операнда из **ST(1)** и < π по абсолютной величине.

Может выполняться над любыми операндами кроме NAN, давая результаты для различных нулей и бесконечностей, как это определено в стандарте IEEE 754.

Инструкция **FPATAN** возвращает угол между осью X и линией от начала координат до точки (X, Y), где Y (ордината) — это **ST(1)**, а X (абсцисса) — это **ST(0)**. Угол зависит от знака X и Y независимо, а не только от знака отношения Y/X. Это связано с тем, что точка (-X, Y) находится во втором квадранте, в результате получается угол между π/2 и π, в то время как точка (X, -Y) находится в четвертом квадранте, в результате получается угол между 0 и −π/2. Точка (−X, −Y) находится в третьем квадранте, что дает угол между −π/2 и −π.

В мануале Intel приведена таблица

		ST(0)						
		− ∞	− F	− 0	+ 0	+ F	+ ∞	NaN
ST(1)	− ∞	− 3π/4*	− π/2	− π/2	− π/2	− π/2	− π/4*	NaN
	− F	-p	−π to −π/2	−π/2	−π/2	−π/2 to −0	- 0	NaN
	− 0	-p	-p	-p*	− 0*	− 0	− 0	NaN
	+ 0	+p	+ p	+ π*	+ 0*	+ 0	+ 0	NaN
	+ F	+p	+π to +π/2	+ π/2	+π/2	+π/2 to +0	+ 0	NaN
	+ ∞	+3π/4*	+π/2	+π/2	+π/2	+ π/2	+ π/4*	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

F2XM1 — вычислить $2^x - 1$

Возводит 2 в степень, заданную в **ST(0)** и вычитает 1.

Результат сохраняется в **ST(0)**.

Значение в **ST(0)** должно лежать в пределах от -1 до +1, иначе результат не определен.

Значения, отличные от двух, могут быть возведены в степень с использованием следующей формулы:

$$x^y = 2^{(y \cdot \log_2 x)}$$

FYL2X — вычислить $y \times \log_2 x$

Вычисляет ST(1) $\times \log_2$(ST(0)), помещает результат в **ST(1)** и выталкивает **ST(0)**.

После этой операции результат оказывается в **ST(0)**.

Первоначальное значение **ST(0)** должно быть неотрицательным.

Если **ST(0)** содержал 0, результатом будет (если **ZM = 1**) бесконечность со знаком, обратным знаку **ST(1)**.

Инструкция **FYL2X** со встроенным умножением разработана для оптимизации вычисления логарифмов с произвольным положительным основанием (b):

$$\log_b x = (\log_2 b)^{-1} \cdot \log_2 x$$

FYL2XP1 — вычислить $y \times \log_2(x+1)$

Вычисляет ST(1) $\times \log_2$(ST(0) + 1), помещает результат в **ST(1)** и выталкивает **ST(0)**.

После этой операции результат оказывается в **ST(0)**.

Первоначальное значение **ST(0)** должно быть в пределах от $-(1-\sqrt{2}/2)$ до $(1-\sqrt{2}/2)$, в противном случае результат не определен.

Команда дает большую точность для **ST(0)** близких к нулю, чем **FYL2X** для суммы того же **ST(0)** и 1.

Выражение (ε + 1) обычно используется при расчетах сложных процентов и сложной ренты. Результат можно просто преобразовать в значение с другим основанием логарифма, включив масштабный коэффициент в исходный операнд **ST(1)**.

Для вычисления масштабного коэффициента **k** при использовании конкретного основания логарифма, требуемое для результата инструкции **FYL2XP1**, используется следующее выражение, где **n** — основание логарифма:

$$k = \log_n 2$$

Команды управления FPU

`WAIT, FWAIT` — ожидание готовности FPU

Процессор проверяет наличие необработанных и незамаскированных исключений и обрабатывает их. Команда указывается в критических ситуациях после команд, чтобы убедиться, что возможные исключения будут обработаны.

`FINIT/FNINIT` — — инициализация FPU/инициализация FPU без ожидания.

Команды устанавливают значения по умолчанию в регистрах **CR**, **SW**, **TW**, **FIP**, **FDP**.
Управляющий регистр инициализируется значением 037h
- округление к ближайшему значению;
- 64-битная точность;
- все исключения замаскированы.

Регистр состояния обнуляется
- TOP = 0;
- флаги исключений сбрасываются.

Регистры данных не меняются, но помечаются как пустые в регистре **TW**.
Регистры **FIP** и **FDP** обнуляются.
FINIT в отличие от **FNINIT** проверяет наличие произошедших и необработанных исключений и обрабатывает их до инициализации. **`FINIT`** полностью эквивалентна **`WAIT FNINIT`**.

FINCSTP — инкремент указателя вершины стека.

Поле **TOP** регистра состояния **SW** увеличивается на 1 по модулю 8. Если **TOP** было равно 7, оно становится равным 0.

Эта команда не эквивалентна выталкиванию **ST(0)** из стека, потому что регистр, который назывался **ST(0)** и стал **ST(7)** не помечается как пустой.

C1 = 0, C0, C2, C3 не определены

FDECSTP — декремент указателя вершины стека.

Поле **TOP** регистра состояния **SW** уменьшается на 1 по модулю 8. Если **TOP** было равно 0, оно становится равным 7.

Содержимое регистров данных и **TW** не изменяется.

C1 = 0, C0, C2, C3 не определены

FFREE op — освободить регистр данных.

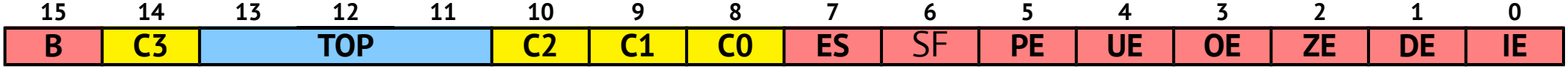
Команда отмечает в регистре **TW**, что операнд (регистр **ST(n)**) пустой. Содержимое регистра и **TOP** не изменяется.

C0, C1, C2, C3 не определены

FCLEX — обнулить флаги исключений

FNCLEX — обнулить флаги исключений без ожидания.

Регистр состояний SR



Команды обнуляют флаги исключений **PE**, **UE**, **OE**, **ZE**, **DE**, **,** а также флаги **ES** (ошибка FPU), **SF** (ошибка стека FPU) и **B** (FPU занят) в регистре состояния FPU.

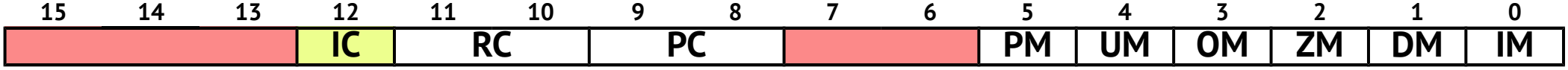
C0, C1, C2, C3 не определены

FCLEX в отличие от **FNCLEX** проверяет наличие произошедших и необработанных исключений и обрабатывает их до инициализации.

FCLEX полностью эквивалентна **WAIT FNCLEX**.

FSTCW m2byte — сохранить регистр CR
FNSTCW m2byte — сохранить регистр CR без ожидания.

Регистр управления CR



Команды копируют содержимое регистра CR в приемник (16-разрядная переменная).
C0, C1, C2, C3 не определены.

FSTCW в отличие от **FNSTCW** проверяет наличие произошедших и необработанных исключений и обрабатывает их до инициализации.
FSTCW полностью эквивалентна **WAIT FNSTCW**.

FLDCW m2byte — загрузить регистр CR.

Копирует содержимое источника (16-бит память) в регистр CR.
Если в регистре **SR** установлены и замаскированы в **CR** один или несколько флагов исключений, а команда **FLDCW** эти маски удалила, перед началом выполнения следующей команды FPU (кроме команд без ожидания) будут обработаны соответствующие исключения.
Чтобы этого не происходило, перед **FLDCW** выполняют **FCLEX** (очистка флагов исключений).
C0, C1, C2, C3 не определены.

FSTENV **m14/28byte** — сохранить вспомогательные регистры
FNSTENV **m14/28byte** — сохранить вспомогательные регистры без ожидания

Сохраняет все вспомогательные регистры FPU в приемник (14 или 28 байт в памяти в зависимости от разрядности операндов).

Маскирует все исключения.

Сохраняет содержимое регистров **CR**, **SR**, **TW**, **FIP**, **FDP** и последнюю команду в формате, зависящем от текущей разрядности операндов и адресов (7 двойных слов для 32-битных операндов, 7 слов для 16-битных операндов).

```
DEST[FPUControlWord]           := FPUControlWord;
DEST[FPUStatusWord]            := FPUStatusWord;
DEST[FPUTagWord]               := FPUTagWord;
DEST[FPUDataPointer]           := FPUDataPointer;
DEST[FPUInstructionPointer]    := FPUInstructionPointer;
DEST[FPULastInstructionOpcode] := FPULastInstructionOpcode;
```

Первое слово (младшая половина первого двойного слова) содержит **CR**, второе — **SR**, третье — **TW**, четвертое — **FIP**. Содержимое пятого, шестого и седьмого слов варьируется в зависимости от текущей разрядности адресации и операндов.

Детали — в документации.

FSTENV и **FNSTENV** различаются аналогично другим командам с ожиданием.

C0, C1, C2, C3 не определены.

FLDENV m14/28byte — загрузить вспомогательные регистры.

Загружает все вспомогательные регистры FPU (**CR**, **SR**, **TW**, **FIP**, **FDP**) из источника (область памяти размером 14 или 28 байт в зависимости от разрядности операндов), сохраненные ранее командой **FSTENV**/**FNSTENV**.

```
FPUControlWord           := SRC[FPUControlWord];
FPUStatusWord            := SRC[FPUStatusWord];
FPUTagWord               := SRC[FPUTagWord];
FPUDataPointer           := SRC[FPUDataPointer];
FPUInstructionPointer    := SRC[FPUInstructionPointer];
FPULastInstructionOpcode := SRC[FPULastInstructionOpcode];
```

Если в загружаемом **SW** установлены и одновременно не замаскированы флагами в **CR** одно или несколько флагов исключений, перед началом выполнения следующей команды FPU (кроме команд без ожидания) будут выполнены исключения.

C0, C1, C2, C3 будут загружены сохраненными в **src** значениями.

FSAVE m94/108byte — сохранить состояние FPU
FNSAVE m94/108byte — сохранить состояние FPU без ожидания

Сохраняет состояние FPU (регистры данных и вспомогательные регистры) в приемник (область памяти размером 94 или 108 байт в зависимости от разрядности операндов) и инициализирует FPU аналогично командам **FINIT**/**FNINIT**.

FSAVE в отличие от **FNSAVE** проверяет наличие произошедших и необработанных исключений и обрабатывает их до выполнения.

FSAVE полностью эквивалентна **WAIT FNSAVE**.

Используется для смены контекста FPU при переключении задач.

Что, в какой последовательности и куда сохраняется смотрим в документации.

C0, C1, C2, C3 будут сохранены, после чего очищены

FRSTOR m94/108byte — восстановление состояния FPU.

Загружает состояние FPU (регистры данных и вспомогательные регистры) из источника (область памяти размером 94 или 108 байт в зависимости от разрядности операндов), который был заполнен командой **FSAVE/FNSAVE**.

C0, C1, C2, C3 будут загружены.

FXSAVE m512byte — быстрое сохранение состояния FPU

Сохраняет текущее состояние FPU, включая все регистры, в приемник (512-байтную область памяти с адресом, кратным 16), не проверяя на необработанные исключения, аналогично команде **FNSAVE**.

Что, в какой последовательности и куда сохраняется смотрим в документации.

48 байт (464:511) в области сохранения доступны для нужд программы.

В отличие от **FNSAVE** инструкция **FXSAVE** не инициализирует FPU после сохранения состояния.

Инструкция FXSAVE не совместима с FRSTOR.

FXRSTOR m512byte — быстрое восстановление состояния FPU.

Загружает состояние FPU, включая все регистры, из источника (512-байтная область памяти с адресом, кратным 16), который был заполнен командой **FXSAVE**.

(регистры данных и вспомогательные регистры) из источника (область памяти размером 94 или 108 байт в зависимости от разрядности операндов).

FSTSW m2byte/AX — сохранить регистр **SR**
FNSTSW m2byte/AX — сохранить регистр **SR** без ожидания

Сохраняет текущее значение регистра **SR** в приемник (регистр AX или 16-битная переменная).

Команда **FSTSW AX** обычно используется после команд сравнения, команд **FPREM, FPREM1** и **FXAM** для пересылки флагов **C3**, **C2**, **C0** в регистр флагов CPU с тем, чтобы выполнить условный переход.

FNOP — занимает место и время, но не выполняет никаких действий.