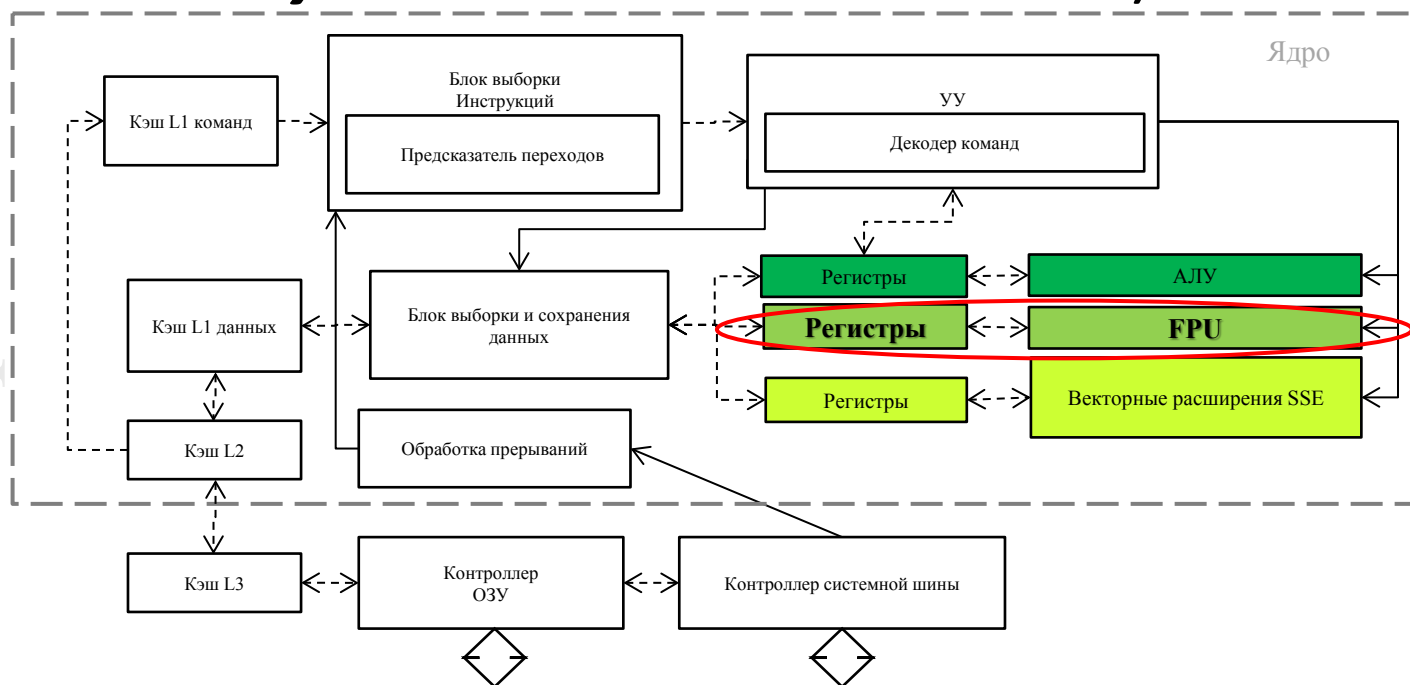


Арифметический процессор FPU

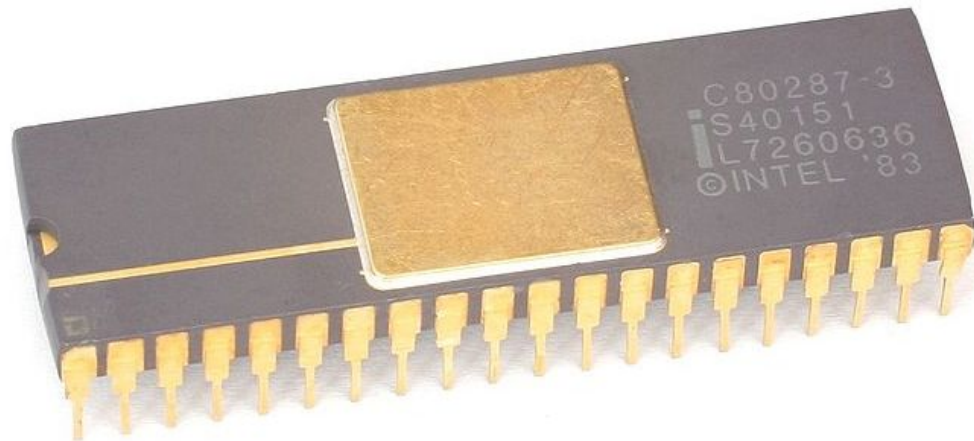
Арифметический процессор FPU

- **FPU** - Floating Point Unit
- Отдельное операционное устройство для операций над числами с *плавающей точкой* со своими РОН и системой команд (частично поддерживает *целые двоичные* и *упакованные BCD числа*)



Арифметический сопроцессор

- Первоначально был отдельным **сoproцессором** (чипом **x87**) – 8087...80487, позже интегрирован в ядро процессора x86 (с Pentium)

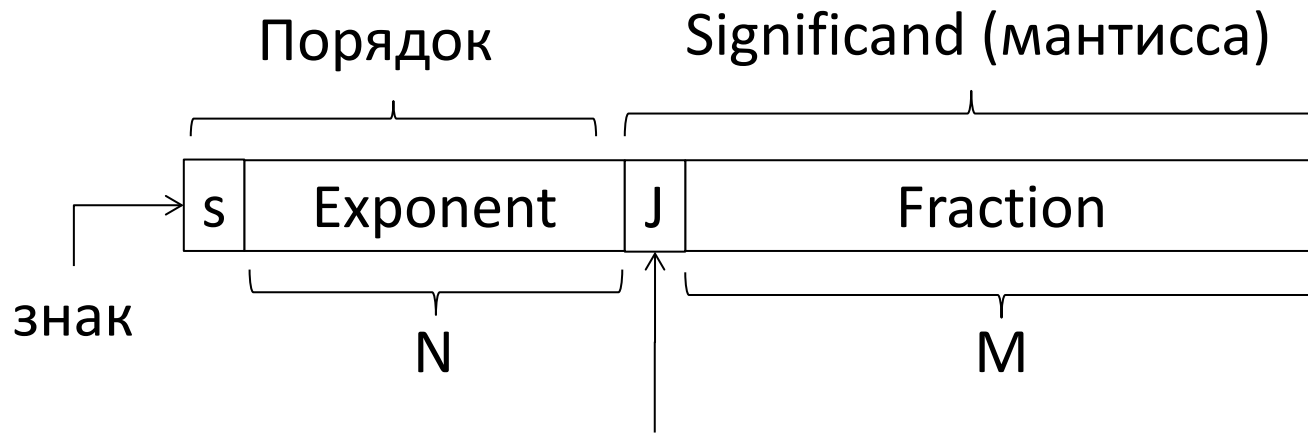


- Теперь **x87** – подмножество инструкций семейства **x86**

Применение FPU

- Для **простых** действий над **вещественными** числами эффективнее SIMD- процессор, расширения SSE, AVX
- Но - FPU:
 - обеспечивает большую точность (за счет хранения **80-битных** промежуточных результатов)
 - может вычислять транscendentные функции (тригонометрические, логарифмическая, показательная)

FPU - Представление чисел с плавающей точкой



J-бит = 1 – целая часть мантиссы

формат	длина	N	M	J-бит
short real (single precision)	32	8	23	нет
long real (double pr.)	64	11	52	
extended real (extended pr.)	80	15	63	есть

Представление чисел с плавающей точкой

- **Экспонента** записана **без знака**
(относительный, смещенный порядок!)
- **Значение числа =**

$$\text{знак} * 2^{\text{Exponent} - \text{bias}} * (1. \text{Fraction})$$

где $\text{bias} = 2^{N-1} - 1$ – смещение порядка

- **Целая единица (J-bit) в 32 и 64-бит не записывается (но подразумевается)**

Пример

- Тип – single precision

1	1000 0001	010 0000	...	0000
---	-----------	----------	-----	------

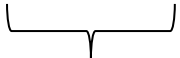
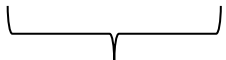
$$-1 * 2^{129-127} * (1.01_2) = -4 * 1.25 = -5$$

Значения вещественных чисел

число, смысл	экспонента	мантисса
нормализованное	$\neq 0$	Любая ($J=1$)
денормализованное	0	$J=0, \underbrace{*** \dots **}_{\text{не все нули}}$
± 0	0	0
$\pm \infty$	111...11	0

0 и ∞ имеют знак!

Значения вещественных чисел

число, смысл	экспонента	мантисса
SNaN (Signal Not a Number) сигнальное не число	111...11	J=1, 0**...**  не все нули
QNaN (Quiet ...) тихое не число	111...11	J=1, ***...**  любые
неподдерживаемое	остальные случаи	

QNaN

- Является результатом:
 - операций с NaN
 - многих недействительных операций
- Если является аргументом, не вызывает исключения
- Обеспечивает “распространение ошибки”

SNaN

- Если является аргументом, вызывает исключение
- Может присваиваться переменным для обнаружения ошибок, связанных с доступом к недействительным данным

Что это за числа?

- Тип – single precision

1	1000 0000	111 1111 1111 1111 1111 1111
---	-----------	------------------------------

обычное число

0	1111 1111	000 0000 0000 0000 0000 0000
---	-----------	------------------------------

$+\infty$

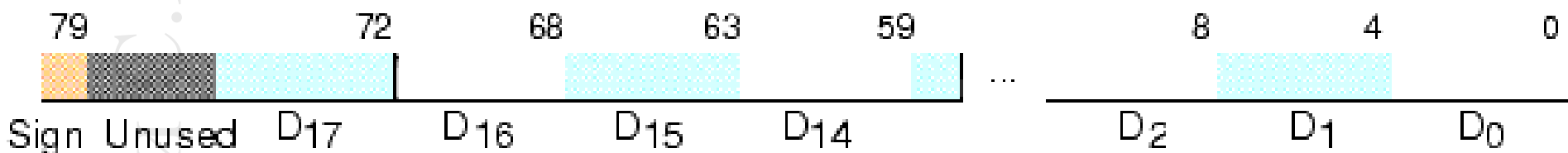
1	1111 1111	111 1111 1111 1111 1111 1111
---	-----------	------------------------------

– NaN

Поддержка целых чисел

Может загружать и сохранять:

- **Целые знаковые числа 16, 32, 64 бита**
- **Упакованное BCD число длиной 80 бит**
 - 18 цифр в 9 младших байтах
 - знак в старшем бите



- После загрузки целые числа преобразуются в вещественные!

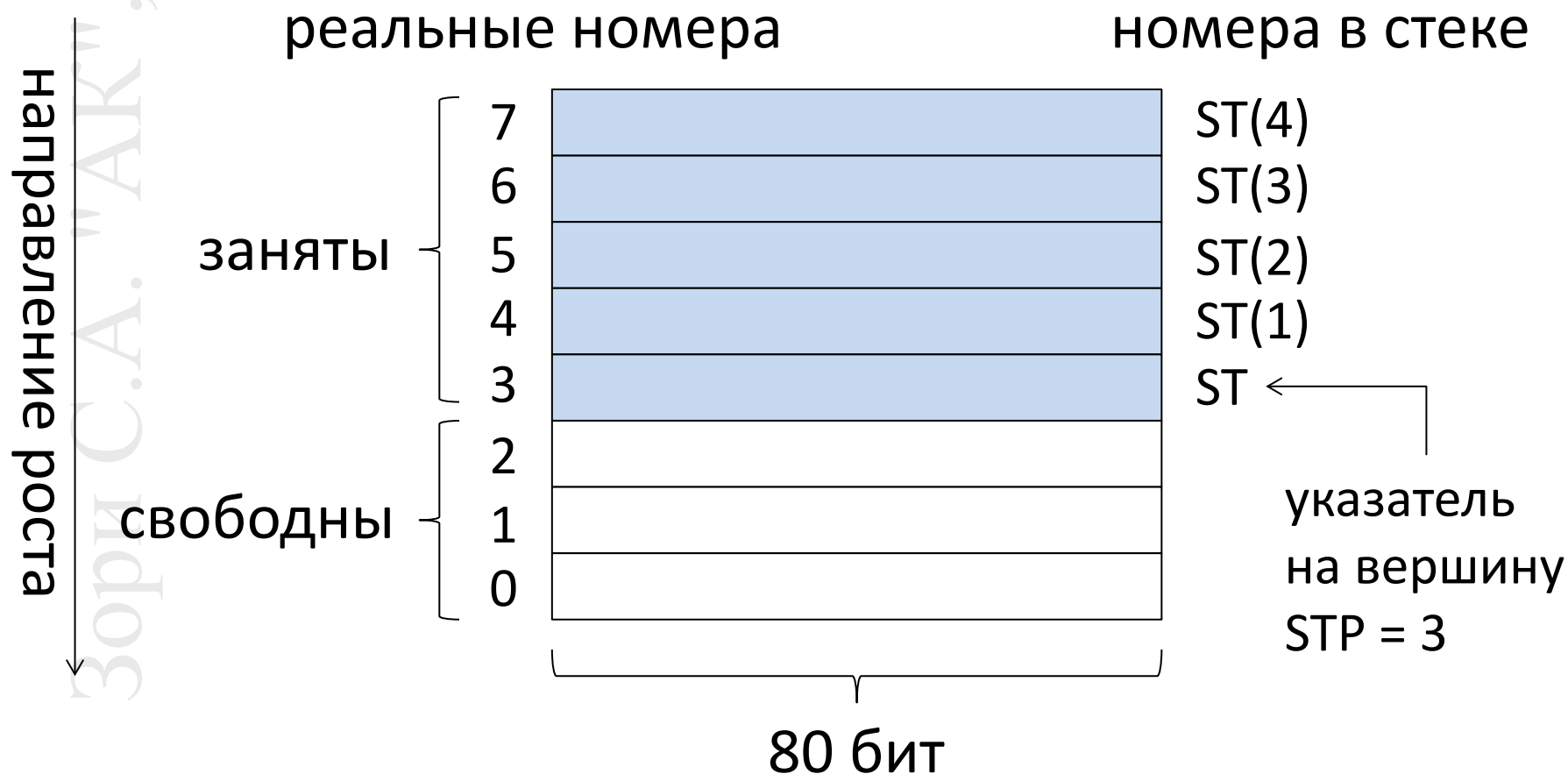
Регистры FPU - Стек регистров

Механизм работы FPU основан на **Стековой машине с ОПН!**

В связи с этим, **регистры** (*собственная память FPU*) организованы в виде **Стека регистров**, а **выполнение (*порядок*) вычислительных действий FPU полностью определяется ОПН** (*фактически, заменив каждое необходимое действие в записи ОПН командой FPU, получаем готовую программу!*).

Регистры FPU - Стек регистров

- 8 регистров для хранения данных



Стек регистров FPU

- Программный доступ не по абсолютным номерам, а по позиции в стеке
- Вершина стека обозначается ST или ST(0), предыдущие регистры – ST(1), ST(2), ...
- Физический номер регистра =
(STP + номер в стеке) по модулю 8

Вспомогательные регистры FPU (5)

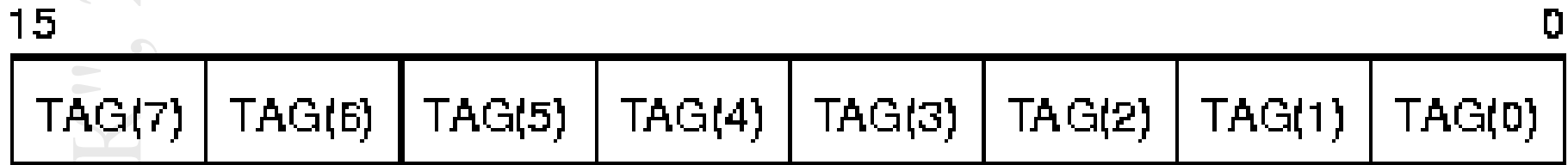
- **CR** (Control Register, 16 бит) – битовые поля, которые можно изменять для управления работой FPU
- **SW** (Status Word, 16 бит) – битовые поля, показывающие статус FPU
- **TW** (Tag Word, 16 бит) – хранит тип содержимого каждого регистра стека

Вспомогательные регистры FPU (5)

- **FIP** (FPU Instruction Pointer, 48 бит) и **FDP** (FPU Data Pointer, 48 бит)
 - указывают на последнюю выполненную инструкцию (кроме контрольных – **FINIT**, **FSTSW**, ...) и ее операнд
 - используются обработчиками исключений

Регистр TW

- На каждый регистр стека – 2 бита



- **Значения:**

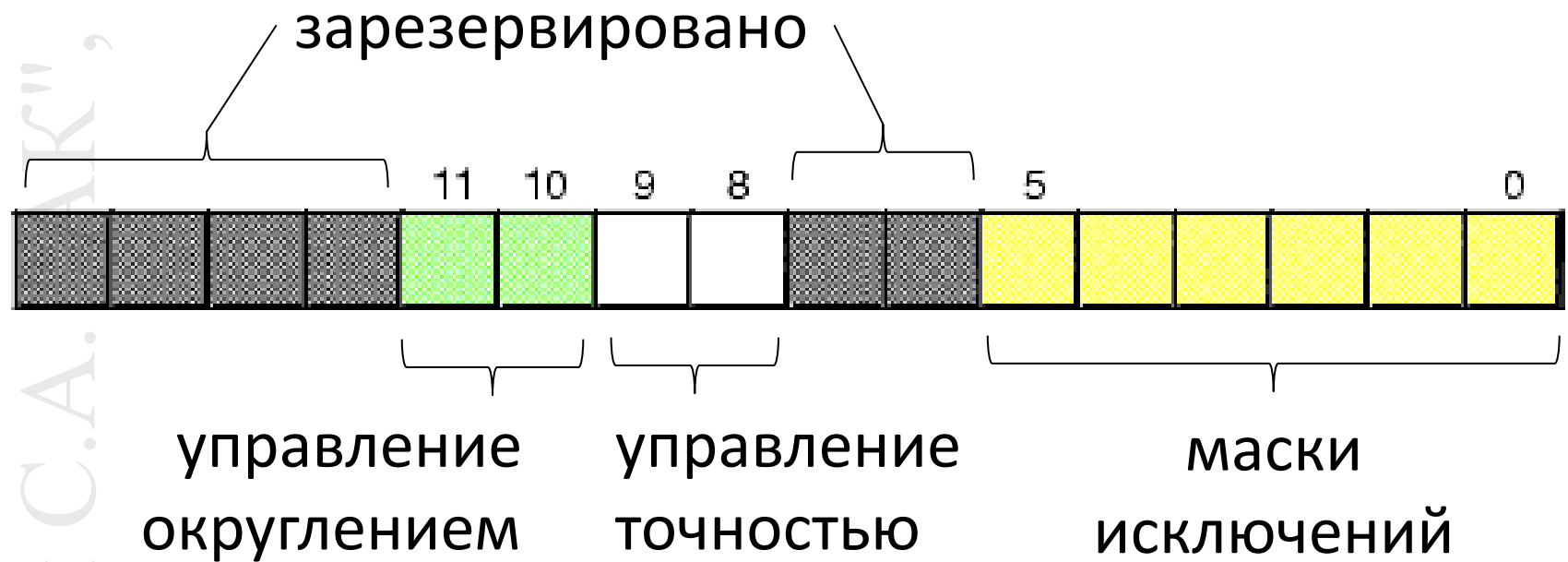
00 – обычное число

01 – ноль

10 – особое (NaN, неподдерживаемое, бесконечность, денормализованное)

11 – пусто

Регистр CR



Регистр CR

- Управление точностью – для совместимости со старыми программами по стандарту IEEE 754.
- В новых программах должно быть значение 11 для максимальной точности (=64 бит).

Регистр CR

- Управление округлением – когда результат вычислений нельзя представить точно или он приводится к меньшей точности (64/32 бита)

значение	округление
00	к ближайшему или четному
01	Вверх
10	вниз
11	обрезка

Регистр CR

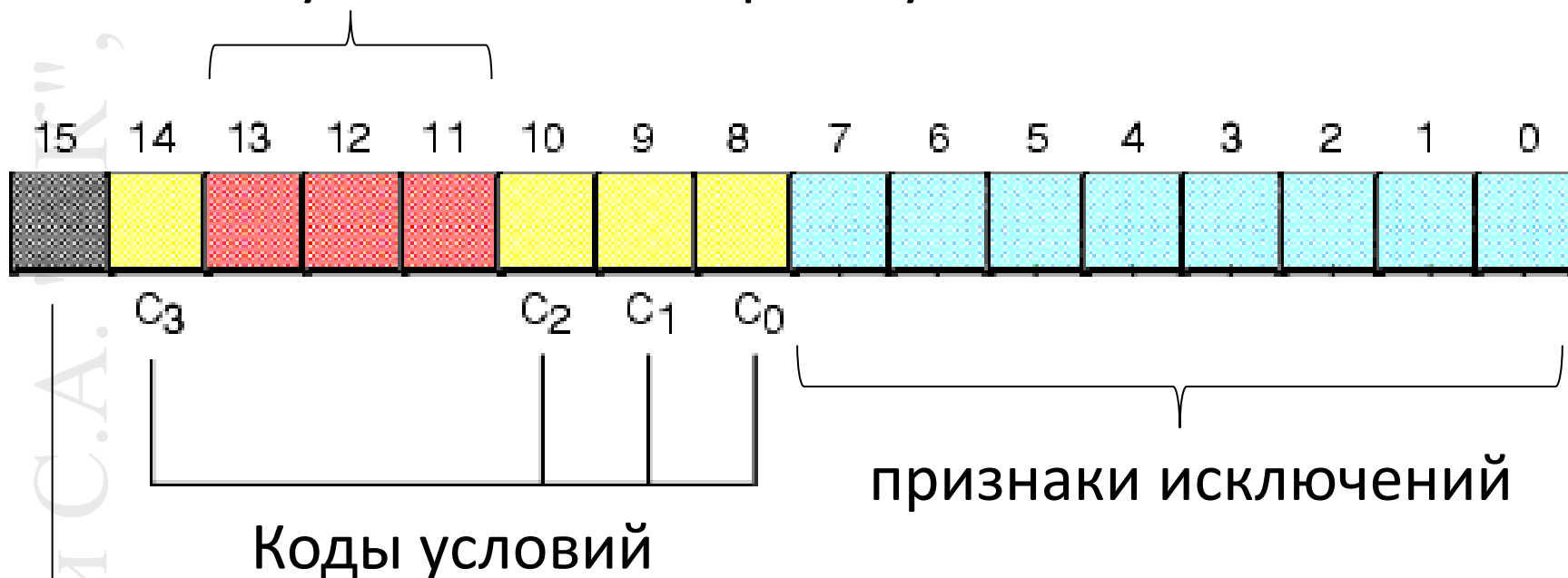
- 00 – значение по умолчанию, обычно дающее наилучшую точность для последовательностей операций

отбрасываемые биты	округление
от 000...00 до 011...11	вниз
100...00	к четному
от 100...01 до 111...11	вверх

Регистр SW

STP (Stack top pointer)

указатель на вершину стека



флаг, показывающий, что FPU занят

Обработка исключений FPU

- Существует 6 типов исключений, пронумерованных от 0 до 5
- При возникновении N-го типа исключения устанавливается:
 - N-й бит **SW**
 - 7-й бит **SW** (признак любого исключения)

Обработка исключений FPU

- Режим обработки N-го типа исключений определяется N-м битом регистра **CR** (маска исключения)
- маска=0 – не маскированное – вызывается обработчик исключения (int 10h или IRQ 13, что определяется центральным процессором)
 - команды **WAIT**, **FWAIT**: CPU ждет, пока не будут обработаны все незамаскированные исключения FPU

Обработка исключений FPU

- маска=1 – маскированное – выполняется действие по умолчанию, не прерывающее работу программы

Типы исключений FPU

0 – неточный результат

Пример: $1/3$ нельзя точно представить в двоичной системе

ДМИ (Действие для Маскируемых Исключений):

- округлить результат согласно CR
- если произошло округление вверх, установить $C1 = 1$, иначе $C1 = 0$

Типы исключений FPU

1 – антипереполнение

Результат слишком мал, чтобы быть представленным в нормализованном виде

ДМИ:

- денормализованный результат

Типы исключений FPU

2 – переполнение

Результат слишком большой

ДМИ:

- результат преобразуется в ∞ того же знака

Типы исключений FPU

3 – деление на ноль

ДМИ:

- результат преобразуется в ∞ ,
- ее знак = знак делимого * знак нуля

Типы исключений FPU

4 – встречен денормализованный операнд

ДМИ:

- нормализовать и продолжать выполнение

Типы исключений FPU

5 – недействительная операция. Их типы:

- Ошибка стека. ДМИ:
 - бит 6 регистра **SW** = 1
 - **C1** = 1, если переполнение (overflow),
иначе **C1** = 0 (underflow)
- Сохранение пустого регистра
- Обмен (FXCH), если один из операндов пуст

Типы недействительных операций

- Сохранение слишком большого числа как целого
- Сравнение числа с NaN
 - ДМИ: $C0 = C2 = C3 = 1$

Результат – QNaN в случае:

- Операция с неподдерживаемым числом или NaN

Типы недействительных операций

- Сложение ∞ с разным знаком или вычитание содним
- $0 * \infty, 0 / 0, \infty / \infty$
- $\sqrt{x}, \log(x)$ при $x < 0$
- Частичный остаток от деления (FPREM, FPREM1), если делимое ∞ или делитель 0, ДМИ: C2 = 0
- Тригонометрическая операция над ∞ , ДМИ: C2 = 0

Обозначения команд FPU

- Все имена команд FPU начинаются с **F**
 - работающие с целыми числами – на **FI**
 - работающие с BCD – на **FB**

Обозначения команд

- Многие команды имеют версию с суффиксом **P** (от Pop) – они после основного действия удаляют операнд из стека:
 - помечают $ST(0)$ как пустой;
 - $TOP := TOP + 1$
- Суффикс **PP** – аналогично удаляют 2 аргумента из стека

Обозначения команд

- Если команда * имеет 2 версии, начинающиеся на **F** и **FN**, то:
 - **FN*** – не проверяет наличие необработанных (немаскированных) исключений
 - **F*** эквивалентна паре команд
FWAIT
FN*
т.е. перед выполнением ждет, пока не будут обработаны все исключения

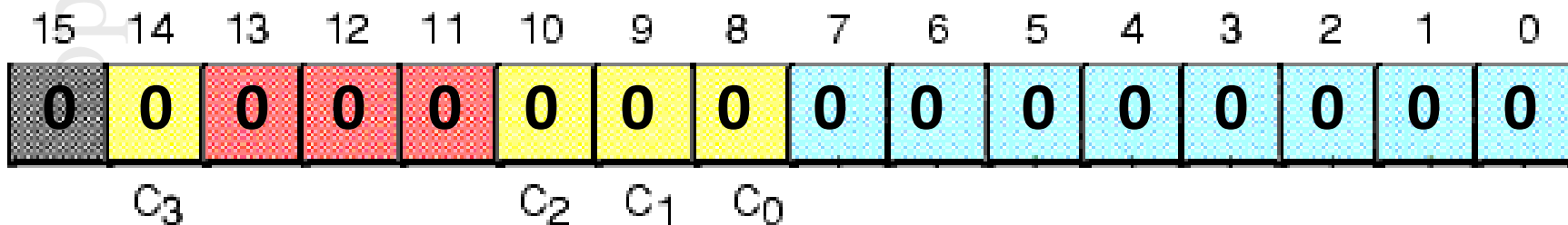
Инициализация FPU

- Команды **FINIT**, **FNINIT** – инициализируют вспомогательные регистры
- $CR := 037Fh$ (округление к ближайшему или четному, макс. точность, все исключения замаскированы)



Инициализация FPU

- (продолжение)
- $TW := FFFFh$ (все регистры свободны)
- $SW := 0$ (STP=0, признаки исключений сброшены)



Работа со вспомогательными регистрами

команда	Действие
FSTCW dst FNSTCW dst	Сохраняет CR в переменную dst
FLDCW src	Загружает CR из переменной src Если сбрасывает признаки исключений – предварительно обрабатывает их
FSTSW dst FNSTCW dst	Сохраняет SW в dst (AX или переменная)
FCLEX FNCLX	Обнуляет флаги исключений

Для FSTCW есть пара FLDCW

Для FSTSW нет пары FLDSW

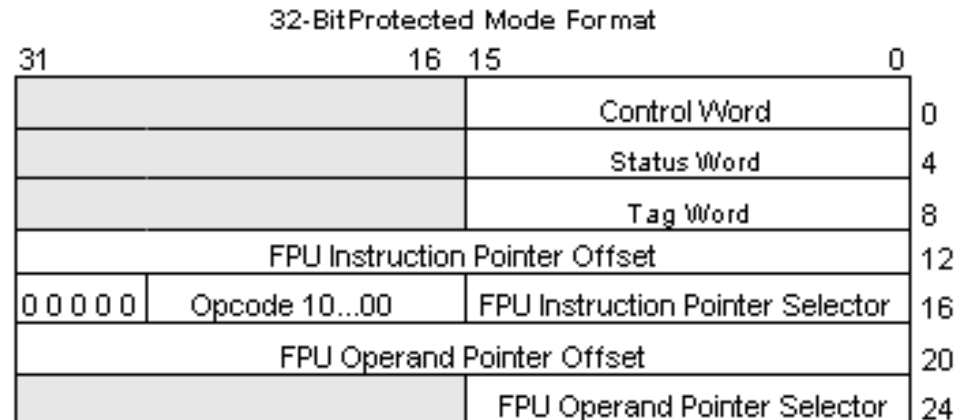
Сохранение состояния FPU

- **Сохранение** некоторых или всех регистров **FPU** в область памяти (14, 28, 94, 108, 512 байт): FSTENV, FNSTENV, FSAVE, FNSAVE, FXSAVE
- **Загрузка** регистров **FPU** из памяти: FLDENV, FRSTOR, FXRSTOR
- Используются при вызове процедур, переключении потоков

Сохранение состояния FPU

FSTENV, FNSTENV (14/28 байт)

- Команды FSTENV и FNSTENV записывают текущую среду FPU в память, а затем маскируют все исключения. Команда FSTENV, в отличие от FNSTENV, проверяет на наличие отложенных необработанных исключений.
- Среда FPU состоит из управляющего слова FPU (**CW**), слова состояния (**SW**), слова тэгов (**TW**) и указателей ошибки для данных и команды (**FDP, FIP**).



For instructions that also store x87 FPU data registers, the eight 80-bit registers (R0-R7) follow the above structure in sequence.

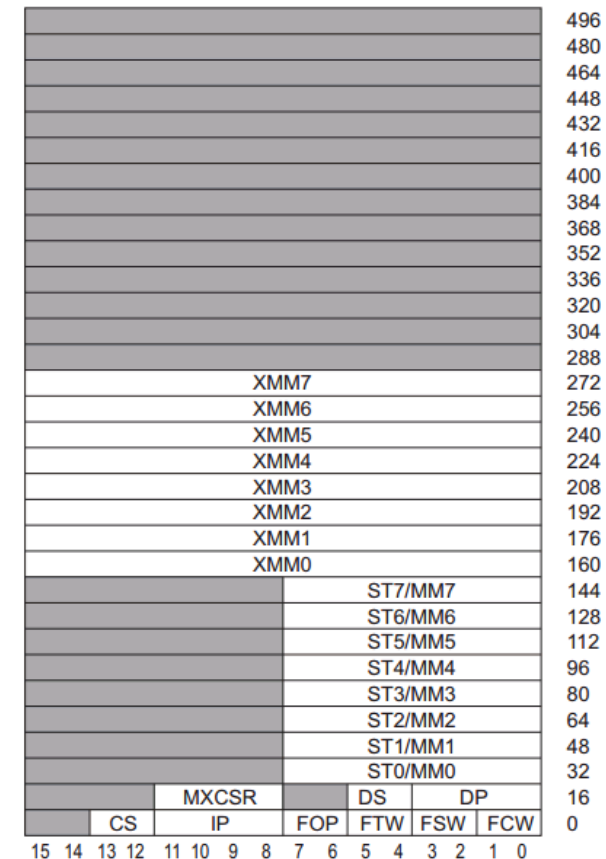
Сохранение состояния FPU

- FSAVE/FNSAVE** (94/108 байт) записывает текущий контекст FPU (среду и регистровый стек) в память, а затем инициализирует FPU аналогично тому, как это происходит при выполнении команд FINIT/FNINIT.
- Команда FSAVE, в отличие от FNSAVE, проверяет на наличие отложенных необработанных исключений.
 - Среда FPU состоит из управляющего слова FPU (CW), слова состояния (SW), слова тэгов (TW) и указателей ошибки для данных и для команды (FDP, FIP) – 28 байт.
 - Регистры стека FPU, начиная с ST(0) и заканчивая ST(7), занимают 80 байт и размещаются сразу же за образом среды.

Сохранение состояния FPU

FXSAVE (512 байт) осуществляет сохранение в области памяти содержимого всех регистров FPU/MMX и SIMD.

- Эти данные могут затем загружаться обратно в процессор командой FXRSTOR.



■ Зарезервировано.