1. Какие регистры используются в сопроцессоре для хранения операндов?

Восемь регистров данных, согласно документации Intel [15], носят имена r_0 – r7, но обратиться к ним по этим именам невозможно. Они образуют стек с плавающей вершиной, построенный по принципу кольцевого буфера.

К регистру, находящемуся сейчас в вершине стека, можно обратиться как к st(0); если стек содержит более одного элемента, то к более глубоким элементам можно обращаться по именам st(1), st(2) и так далее до st(7)

Регистры данных сопроцессора хранят вещественные числа в 80-битном расширенном формате. Мантисса занимает 64 бита, порядок — 15 бит, под знак отводится один бит.

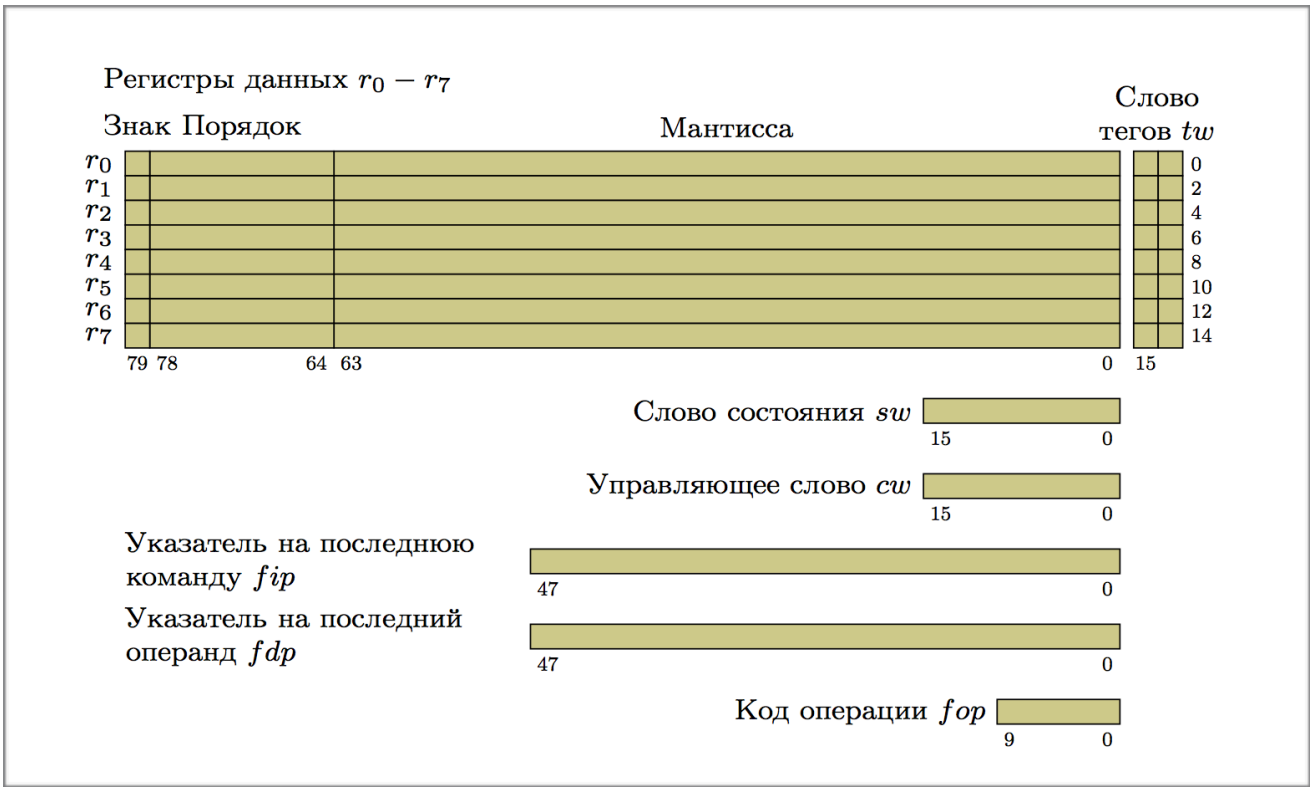

Рис. 5.1. Регистры FPU

Шестнадцатибитный регистр (слово) тегов *tw* (Tag Word, также используется сокращение *twr*— Tag Word Register) хранит состояние регистров данных. Каждому регистру s_0 – s_7 соответствует два бита слова тегов (рис. 5.2):

а) 00 — в соответствующем регистре корректное ненулевое значение;

б) 01 — в регистре ноль;

в) 10 — в регистре специальное значение: некорректное значение (NaN или значение, не соответствующее формату вещественного числа с расширенной точностью), бесконечность или денормализованное число;

г) 11 — регистр пуст.

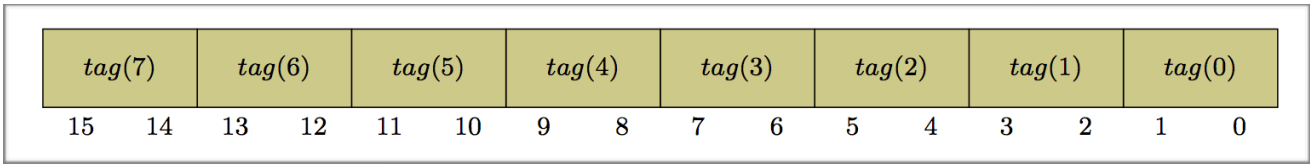

Рис. 5.2. Слово тегов FPU

Если регистр помечен в слове тегов как пустой, его значение при этом может быть каким угодно — попытка чтения из него приведёт к ошибке стека.

Флаги математического сопроцессора разбиты на два шестнадцатибитных регистра (рис. 5.3) — управляющие флаги составляют управляющее слово cw (Control Word, также cwr), флаги состояния сгруппированы в слово состояния sw — (Status Word, также swr).

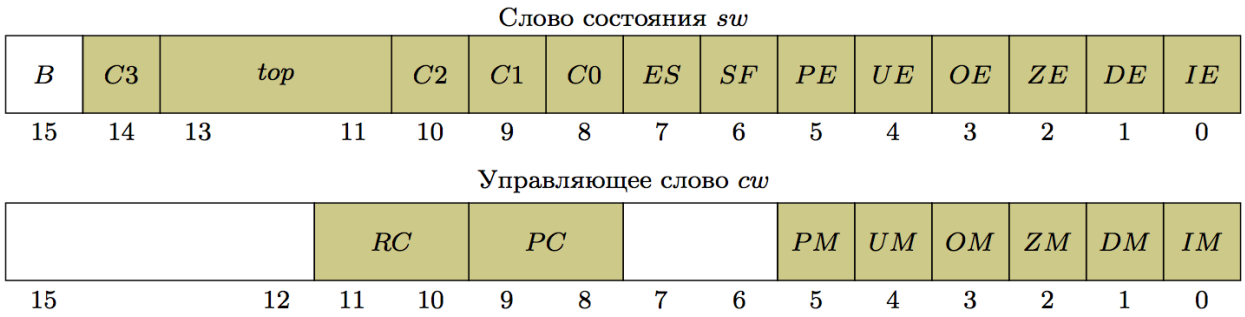

Рис. 5.3. Слово состояния и управляющее слово FPU

Управляющее слово содержит шесть масок исключений (IM−PM), поле управления точностью PC, и поле управления округлением RC.

Слово состояния отображает текущее состояние сопроцессора после выполнения последней команды. Младший байт слова состояния включает семь флагов, показывающих корректность операций (IE − SF) и флаг ES, показывающий, что сбой не только был, но и привёл к прерыванию. Старший байт включает флаги C0 − C3, хранящие результаты сравнения чисел, а также трёхбитный текущий номер вершины стека top . Последний бит B в настоящее время не используется.

ИТОГИ :

1) Таким образом, стек сопроцессора организован с помощью восьми регистров данных $r_0 - r_7$, соответствующих восьми полей слова тегов tag(0)− tag(7) и поля top слова состояния.

2) Вершина стека st(0) находится в регистре r(top), обозначение st(1) получает следующий регистр r(top+1) и так далее. За r_7 по принципу кольцевого буфера следует r_0. На рис. 5.4 показаны соотношения между физическими r(i) и логическими st(i) именами регистров данных сопроцессора при различных значениях номера вершины стека top.

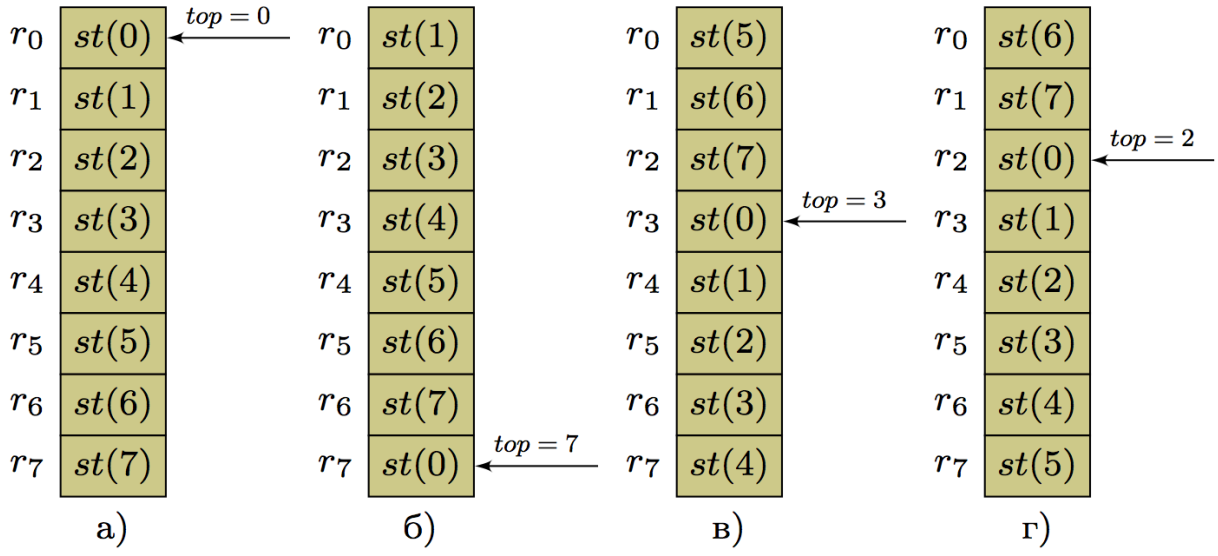

Рис. 5.4. Стек FPU

3) Положение дна стека определяется словом тегов (первый пустой регистр).

- После инициализации стек пуст.
- После завершения вычислений (перед выходом из функции или ассемблерной вставки) его также необходимо оставить пустым.
- Если функция возвращает вещественное значение через стек сопроцессора, в стеке не должно остаться ничего, кроме возвращаемого значения.
- Для вычислений хотя бы один операнд должен быть загружен в стек сопроцессора.

4) Два 48-битных регистра указателей (на последнюю команду—FPU Instruction Pointer fip, , в некоторых источниках также ipr и последний загруженный операнд Data (Operand) Pointer, fdp, также dpr), а также десятибитный регистр кода операции последней неуправляющей команды (FPU Opcode Register, fop) используются в обработке исключений для определения места сбоя.

2. Какие команды используются для выполнения арифметических операций над вещественными числами?

Все мнемонические обозначения начинаются с символа f (FPU). Вторая буква мнемонического обозначения определяет тип операнда в памяти, с которым работает команда:

- i — целое двоичное число со знаком;
- b — целое двоично-десятичное (BCD) число;
- отсутствие буквы — вещественное число.

Последняя буква p в мнемоническом обозначении команды означает, что последним действием команды обязательно является извлечение операнда из стека (удвоенная pp — из стека извлекаются оба операнда).

Команды FPU не могут иметь непосредственных операндов или операндов-регистров основного процессора.

Если не указано иное, используются следующие обозначения. Операнд-приёмник может быть обозначен либо как dest, если он может быть регистром сопроцессора или переменной в памяти либо как dmem, если он может быть только в памяти. Операнд-источник может быть обозначен как src (регистр сопроцессора или переменная в памяти) или smem(переменная в памяти).

Команды загрузки данных в стек FPU

Таблица 5.13

Команда	**Действие**
`fld src`	Помещает вещественное число *src* на вершину стека сопроцессора Если источник src в памяти, его размер определяется суффиксом команды (по умолчанию — 32 бита, *float*)
Загрузка целых чисел	
`fild smem`	Помещает целое знаковое число *smem* на вершину стека сопроцессора Размер источника определяется суффиксом команды (по умолчанию — 16 бит, *short*)
`fbld smem`	Помещает целое двоично-десятичное число *smem* на вершину стека сопроцессора Размер источника — 80 бит
Загрузка констант	
`fldz`	Загрузка 0
`fld1`	Загрузка 1
`fldpi`	Загрузка π
`fldl2t`	Загрузка $\log_2 10$
`fldl2e`	Загрузка $\log_2 e$
`fldlg2`	Загрузка $\log_{10} 2$
`fldln2`	Загрузка $\ln 2$

В стек можно поместить значение одного из элементов стека сопроцессора, значение из памяти или одну из предопределённого набора констант. Невозможно напрямую загрузить в стек сопроцессора значение регистра основного процессора.

Команды выгрузки данных из стека FPU

Таблица 5.14

Команда	Действие
`fst dest` `fstp dest`	Копирует $st(0)$ в *dest* Выталкивает $st(0)$ в *dest* Приёмник *dest* может быть переменной в памяти или пустым регистром сопроцессора Если *dest* в памяти, его размер определяется суффиксом команды (по умолчанию — 32 бита, *float*)
Выгрузка с округлением	
`fist dmem` `fistp dmem`	Копирует $st(0)$ в *dmem* как целое Выталкивает $st(0)$ в *dmem* как целое Размер приёмника определяется суффиксом команды (по умолчанию — 16 бит, *short*)
`fbst dmem` `fbstp dmem`	Копирует $st(0)$ в *dmem* как двоично-десятичное целое Выталкивает $st(0)$ в *dmem* как двоично-десятичное целое Размер приёмника — 80 бит

Арифметические операции

Основные арифметические операции FPU

Таблица 5.18

Команда	Действие Intel	Действие GAS*
`fadd` (сложение)	$dest = dest + src = st(0) + \xi$	
`fsub` (вычитание)	$dest = dest - src$	$dest = st(0) - \xi$
`fsubr` (обратное вычитание)	$dest = src - dest$	$dest = \xi - st(0)$
`fmul` (умножение)	$dest = dest \cdot src = st(0) \cdot \xi$	
`fdiv` (деление)	$dest = dest/src$	$dest = st(0)/\xi$
`fdivr` (обратное деление)	$dest = src/dest$	$dest = \xi/st(0)$

* ξ — операнд, не лежащий на вершине стека.

Может быть как источником, так и приёмником, в зависимости от используемой формы.

Поведение Intel и GAS совпадает в тех случаях, когда приёмником является s(0), в том числе в ситуациях, когда источник находится в памяти. Также поведение Intel и GAS полностью совпадает для симметричных операций — сложения и умножения.

Поведение Intel и GAS совпадает в тех случаях, когда приёмником является

Согласно документации Intel (и в ассемблерах с синтаксисом Intel) прямое вычитание `fsub` вычисляет $dest - src$, а обратное `fsubr` — $src - dest$, то есть результаты команд `fsub %st(0), %st(i)` и `fsub %st(i), %st(0)` не только записываются в различные регистры, но и отличаются знаком.

В GAS, в соответствии с традиционным поведением Unix-ассемблеров, `fsub` вычисляет $st(0) - \xi$ даже в том случае, если приёмником является ξ. В частности, команды `fsub %st(0), %st(i)` и `fsub %st(i), %st(0)` вычисляют одно и то же значение, но помещают его в разные регистры. Обратное вычитание `fsubr` вычисляет $\xi - st(0)$.

Таким образом, команде `fsub %st(0), %st(i)` соответствует опкод, который, согласно документации Intel, должен соответствовать команде `fsubr` [54]. Анализ сгенерированного компилятором из коллекции GCC кода это подтверждает. Аналогично ведут себя `fdiv`/`fdivr`. Такое поведение в случае сочетания синтаксиса AT&T и платформы x86 в некоторых источниках описывается как баг GCC [49], но из соображений совместимости с имеющимся кодом меняться не будет.

s(0), в том числе в ситуациях, когда источник находится в памяти. Также поведение Intel и GAS полностью совпадает для симметричных операций - сложения и умножения.

Шесть форм основных арифметических команд FPU

Таблица 5.19

Команда	Действие
`fXXXp`	Источник — $st(0)$, приёмник — $st(1)$, после выполнения источник $st(0)$ извлекается из стека (то есть результат получает обозначение $st(0)$) Данная форма эквивалентна `fXXXp %st(0), %st(1)` (ассемблируется в тот же опкод)
`fXXX smem`	Источник — память (вещественное число), приёмник — $st(0)$, указатель стека не изменяется Размер источника определяется суффиксом команды, может быть одинарной или двойной точности, но не 80-битным (по умолчанию — 32 бита, *float*)
`fiXXX smem`	Источник — память (целое знаковое число), приёмник — $st(0)$, указатель стека не изменяется Размер источника определяется суффиксом команды, может быть 16- или 32-, но не 64-битным (по умолчанию — 16 бит, *short*)
`fXXX %st(i), %st(0)`	Источник — $st(i)$, приёмник — $st(0)$, указатель стека не изменяется
`fXXX %st(0), %st(i)`	Источник — $st(0)$, приёмник — $st(i)$, указатель стека не изменяется
`fXXXp %st(0), %st(i)`	Источник — $st(0)$, приёмник — $st(i)$, после выполнения источник $st(0)$ извлекается из стека (то есть результат получает обозначение $st(i-1)$)

Внимание!

Ассемблер Unix исторически использовал для основных арифметических команд FPU те же мнемонические обозначения, что и предложенные Intel, но другую семантику операндов.

Таким образом, в GAS поведение мнемоник несимметричных операций (`fsub`/`fsubr` и `fdiv`/`fdivr`) качественно иное, чем описанное в документации Intel и учебниках, описывающих синтаксис Intel.

3. Какие команды используются для выполнения тригонометрических операций?

Дополнительные арифметические и трансцендентные команды FPU

Таблица 5.20

Команда	Действие
`fabs`	$st(0) = \lvert st(0) \rvert$
`fsqrt`	$st(0) = \sqrt{st(0)}$
`fptan`	$\begin{cases} st(0) = 1 \\ st(1) = \operatorname{tg}\big(st(0)\big) \end{cases}$ — частичный тангенс $st(0)$
`fsincos`	$\begin{cases} st(0) = \cos\big(st(0)\big) \\ st(1) = \sin\big(st(0)\big) \end{cases}$ `fsincos` выполняется столько же времени, сколько `fsin` или `fcos` (и вдвое меньше раздельного расчёта синуса и косинуса)
`fsin`	$st(0) = \sin\big(st(0)\big)$
`fcos`	$st(0) = \cos\big(st(0)\big)$
`fscale`	$st(0) = st(0) \cdot 2^{\lfloor st(1) \rfloor}$, $st(1)$ остаётся в стеке
`f2xmi`	$st(0) = 2^{st(0)} - 1$ Значение $st(0)$ должно лежать в пределах от -1 до $+1$, иначе результат не определён.
`fyl2x`	$\Big[st(1) \to st(0)\Big] = st(1) \cdot \log_2\big(st(0)\big)$, $st(0) \geqslant 0$ Если регистр $st(0)$ содержал ноль, результат (если ZM $= 1$) будет равен бесконечности со знаком, обратным $st(1)$.
`fyl2xp1`	$\Big[st(1) \to st(0)\Big] = st(1) \cdot \log_2\big(st(0) + 1\big)$, $st(0)$ от $-(1 - \frac{\sqrt{2}}{2}) \approx -0{,}3$ до $(1 + \frac{\sqrt{2}}{2}) \approx 1{,}7$, иначе результат не определён. Команда `fyl2xp1` дает бóльшую точность для $st(0)$, близких к нулю, чем `fyl2x` для суммы того же $st(0)$ и 1.
`fprem1`	$\Big[st(1) \to st(0)\Big] = st(0) \bmod st(1)$ — частичный остаток по IEEE-754 Применяется, в частности, для уменьшения аргументов периодических функций: остаток от деления $3 \cdot \pi + 1.2$ на π равен 1.2.

4. Какие команды используются для сравнения вещественных чисел?

FPU включает несколько семейств команд сравнения вещественных чисел. Все они сравнивают приёмник st(0) с некоторым источником src. По аналогии с командой целочисленного сравнения можно сказать, что анализируется знак разности st(0) − src . Так как приёмником является st(0), поведение команд сравнения не различается для GAS и Intel.

Все команды сравнения вещественных чисел сравнивают вершину стека — приёмник st(0) с другим операндом — источником src

Команды сравнения FPU

Таблица 5.21

Команда	Источник	Флаги	Особенности
`fcom[p[p]]` `fcom[p] src`	$st(1)$ src	$C0, C3, C2$	
`fucom[p[p]]` `fucom[p] src`	$st(1)$ src		Не генерируется исключения при сравнении с nan
`ficom[p] smem`	$smem$		$smem$ — целое число
`ftst`	0		
`fcomi[p] %st(i), %st(0)`	$st(i)$	CF, ZF, PF	
`fucomi[p] %st(i), %st(0)`	$st(i)$		Не генерируется исключения при сравнении с nan

По результатам сравнения (в соответствии со знаком разности st(0)−src) устанавливается значение трёх флагов: отрицательности, нуля и несравнимости. Операнды считаются несравнимыми, если хотя бы один из них — тихое нечисло (обычно вещественная неопределённость nan).

Какие флаги регистра eflags содержат результат сравнения вещественных чисел?

Результат современных команды сравнения, напрямую устанавливающих флаги ZF,CF ,PF и обнуляющих остальные флаги состояния в регистре flags , можно анализировать как результат сравнения беззнаковых целых чисел без дополнительных действий.

Значение флагов при сравнении

Таблица 5.22

Соотношения	**Флаги**		
	Отрицательности $C0/CF$	Нуля $C3/ZF$	Несравнимости $C2/PF$
$st(0) > src \quad st(0) - src > 0$	0	0	0
$st(0) = src \quad st(0) - src = 0$	0	1	0
$st(0) < src \quad st(0) - src < 0$	1	0	0
$st(0)$ и src несравнимы	1	1	1