непредвзятый сравнительный анализ элегантности архитектур различных процессоров

крис касперски ака мыщъх, aka nezumi, no-email

изучение древних языков в первую очередь позволяет освободить мысль от оков слова, которое воспринимается как единственное данное… даже элементарные грамматические упражнения зачастую заставляют учащегося древним языкам освободиться от кажущегося ему единственным способа выражения, почувствовать и увидеть свою мысль настолько многосторонне, насколько ни для кого другого это недоступно.

Александр Гимадеев

введение

Непрекращающиеся "священные" войны по поводу "x86 процессоры – дерьмо, xxx процессоры – rules" неприятны в первую очередь тем, что они унижают и дисквалифицируют x86-программистов в глазах всей остальной программистской общественности. Они лишают новичков уважения к x86-архитектуре.

Причем, подавляющее большинство защитников x86-архитектуры, с представителями других процессорных семейств знакомы в лучшем случае понаслышке, а противники x86-архитектуры (львиную долю которых составляют поклонники PDP-11), склонны ухватываться за отдельные, непринципиальные архитектурные особенности, которые в x86 реализованы несколько иначе, чем в их любимом процессоре. В общем, аргументы обоих сторон носят глубоко необъективный и бездоказательный характер, сводящийся в основном к ругани и наездам в стиле: "…м*не пришлось как-то переносить Форт с PDP-11 на I8086, причем последний я видел впервые… так от архитектуры i80x86 до сих пор тошнит (особенно по сравнению с PDP-11)*", "*господи, до чего трудно было преодолеть рвотный барьер, осваивая после 5 лет работы на PDP-11 это интелевое смоляное чучелко. :(Кто работал на PDP-11, думаю, подтвердит*".

Мыщъх'ем была предпринята попытка если не поставить точку в этом вопросе, то по крайней мере, дать спорящим сторонам свежую пищу для размышлений (как знать, быть может после этого в конференциях вместо реплик "сам дурак" наконец-то зазвучат нормальные технические аргументы). Сразу оговорим, что ниже будут сравниваться исключительно программные модели нескольких, наиболее "культовых" процессоров. В первую очередь, это, конечно, PDP-11 – легендарнейших процессор всех времен и народов, породивший огромное количество клонов (и отечественные кальки К1801 в частности), многие из которых исправно работают и поныне; затем серию процессоров 68K от Motorola, известную в первую очередь по Эплам ранних моделей и едва не ставшую основной для IBM PC. Наконец, для полноты картины, мы рассмотрим процессоры семейства DEC Alpha. Мне могут возразить, что сравнивать Альфу со всеми выше перечисленными процессорами не совсем корректно, поскольку он совсем из другой категории. Именно так! И поэтому это лишь усиливает контраст! (Кроме того, Альфа окутана таким количеством мифов, домыслом и легенд, что близкое знакомство с ней, никому не помешает).

Сравнительный анализ охватывает как ключевые архитектурные концепции, так и индивидуальные непринципиальные архитектурные особенности, такие, как например, наличие в PDP-11 команды обнуления, отсутствующей в x86 и вынуждающей программистов использовать либо пересылку непосредственного нуля, либо логическую операцию "ИЛИ исключающее И", что с одной стороны ничуть не ухудшает технические характеристики программы, но с другой – создает впечатления уродства архитектуры.

система команд

Это ведь действительно так – x86 обладает самой идиотской архитектурой и самым нелепым набором команд из всех ныне существующих

процессоров. Так что желание изучить его досконально кажется мне весьма сомнительным.
V. S. Lugovsky aka Mauhuur

Отличительность особенность x86 является чрезвычайно богатый набор машинных команд, количество которых в старших моделях Pentium'ов достигает пятисот, что значительно превышает количество команд во всех остальных популярных процессорах вместе взятых! Разумеется, само по себе чисто поддерживаемых машинных команд еще ни о чем не говорит, напротив, даже вызывает некоторые сомнения в их элегантности. Вспоминается анекдот десятилетней давности: встречаются как-то представители двух функционально одинаковых заводов, один из которых расположен в СССР, а другой в Японии. Вот японский представитель и говорит "На моем заводе работают восемь человек". А русскому стыдно сказать, что у него работает целая тысяча и он, слегка приукрашивая действительность, скашивает количество работающих до девяти. На следующий день заметно нервничающий японец спрашивает нашего: слушай, я всю ночь не спал, но так и не смог понять: что же у тебя девятый рабочий делает-то?

Действительно, чтобы полноценно программировать на ассемблере x86 нужно очень многому научиться, и лишь немногим программистам удается удержать весь этот набор команд у себя в голове. Постоянным спутником программистов становятся тысячи страниц технической документации и вместо того, чтобы сосредоточиться непосредственно на решаемой задаче, программист вынужден снова и снова штудировать многочисленные руководства, пытаясь разобраться в какую же форму следует облечь свою программистскую мысль, чтобы донести ее до "тупой" машины. Ассемблерные программисты уже улыбнулись? И правильно! Ведь большая часть этих пятисот команд относится к векторной и мультимедийной обработке, т. е. к прямо таки скажем к весьма специфичным областям. Избыточность – она, конечно, карман не тянет (во всяком случае, карман программиста, про производителей процессоров мы намерено не говорим), но вот многословность, межусобная обособленность и косность системы команд уродуют листинги только так! Вот только несколько примеров: отсутствие адресации память – память не позволяет обрабатывать переменные, находящиеся в памяти без пересылки их содержимого во временный регистр (и это при том, что количество регистров общего назначения в x86 процессоров крайне невелико); команды x86 сопроцессора используют свои, – причем крайне примитивные, – способы адресации, а потому и синтаксически, и архитектурно обосабливаются от целочисленных команд, что существенно затрудняет как их изучение, так и работу с ними; ряд машинных команд жестко связан со строго определенными регистрами и за счет этого произвола многие из них практически полностью обессмысливаются (в частности, привязка счетчика цикла к регистру ECX/CX не позволяет реализовывать вложенные циклы). Наконец, раздельная адресации оперативной памяти и портов ввода/вывода…

Процессоры семейства PDP-11 и 68K обходятся значительно меньшим количеством команд, причем удобство их ассемблера было столь велико, что позволяло ему на равных конкурировать с Си и другими языками высокого уровня. Теперь конечно, программистские веяния изменились и ассемблер неожиданно выпал из их внимания, однако, недостатки системы команд x86 процессоров тут совершенно не причем (хотя, некоторые и не разделяют такой точки зрения).

Ограниченный объем журнальной статьи не позволяет мыщъх'у рассказать о достоинствах и недостатках каждого из процессоров во всех подробностях, но, это, собственно, и не нужно. Зачем жевать уже жеванное (мыщъх'и к жвачным животным не относятся!), когда табличный материал грызть намного удобнее! Перелопатив кучу страниц документации, мыщъх отделил зерна от плевел и складировал их посреди своей норы, получив в итоге очень внушительную (но зато легкоусвояемую!) таблицу 1, готовую к непосредственному употреблению (желательно с пивом).

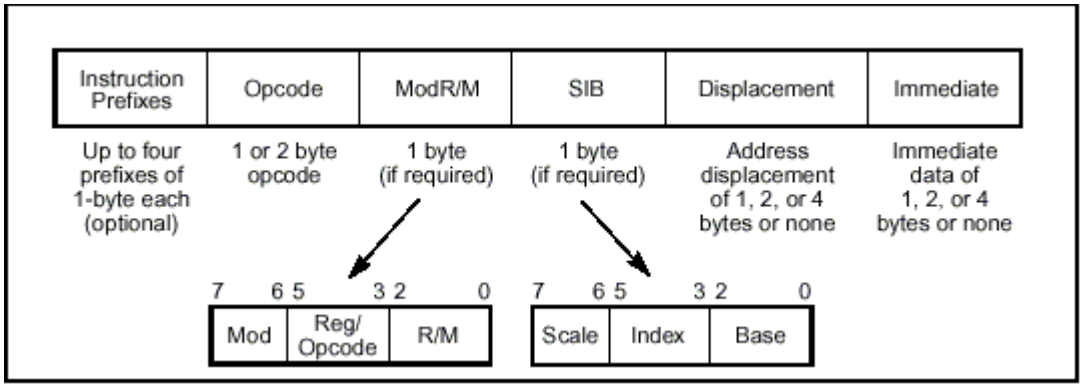

Рисунок 1 формат машинных команд x86-процессоров

```
/*
 *  Hello World in gas/NetBSD
 *  AT/T-Syntax
 *
 *  Compile:
 *    gas hw.s
 *    ld -s -o hw a.out
 */

.data

msg:
        .string "Hello World\n"

len:
        .long . - msg


.text
.globl _start

_start:
        push    $len            /* Laenge */
        push    $msg            /* Adresse */
        push    $1              /* Stdout */
        movl    $0x4, %eax      /* write */
        call    _syscall
        addl    $12, %esp       /* Stack bereinigen */

        push    $0
        movl    $0x1, %eax      /* exit */
        call    _syscall

_syscall:
        int     $0x80
        ret
```

Листинг 1 "Hello World" на x86-ассемблере

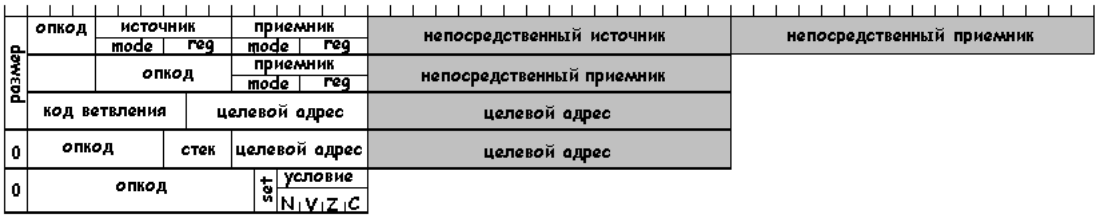

Рисунок 2 формат машинных команд процессоров PDP-11

```
;; Hello world in MIDAS

title   hello
```

```
start:  .open    [.uao,,'tty ? 0 ? 0]
        .lose   %lsfil
        move    1, [440700,,[asciz "Hello, world"]]
loop    ildb    2,1
        skipn   2
.logou  1,
.iot    2
        jrst    loop

end
```

Листинг 2 "Hello, World" на PDP-ассемблере

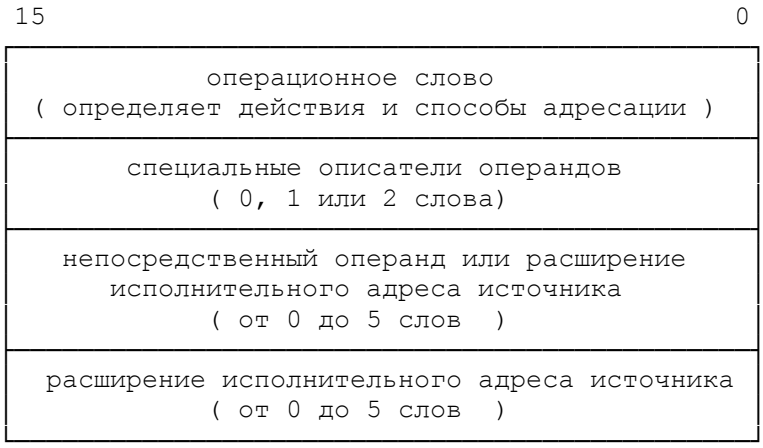

Рисунок 3 формат машинных команд 68k-процессоров

```
start:
        ; Message-String ausgeben
        move.l #msg,-(a7)
        move.w #9,-(a7)
        trap #1
        addq.l #6,a7

        ; auf Taste warten
        move.w #1,-(a7)
        trap #1
        addq.l  #2,a7

        ; Programm beenden
        clr -(a7)
        trap #1

        msg: dc.b "Hello World",10,13,0
```

Листинг 3 "Hello, World" на 68k-ассемблере

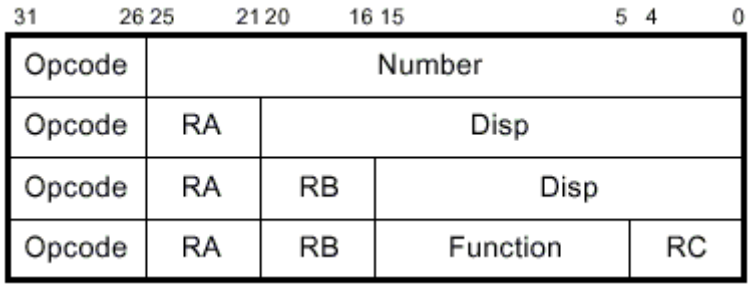

Рисунок 4 формат машинных команд процессора DEC Alpha 21264

>>> врезка сводная таблица сравнительной элегантности программной модели различных процессоров

	x86	PDP	68K	DEC Alpha
основные характеристики				
тип процессора	CISC	CISC	CISC	RISC
система команд				
система команд	безоперандная, одно и двух операндная	безоперандная, одно и двух операндная	безоперандная, одно и двух операндная	безоперандная, одно, двух и **трех** операндная
размер машинной команды	от 1 до 16 байт	1, 2 или 3 слова	от 1 до 12 слов	одно двойное слово
типы команд	пересылки данных	пересылки данных	пересылки данных	пересылки данных
	арифметические	арифметические	арифметические	арифметические
	логические	логические	логические	логические
	управления	управления	управления	управления
	системные	системные	системные	системные
система кодировки машинных команд	синтаксис команд чрезвычайно сложен, – инструкции имеют переменную длину и множество факультативных контекстно-чувствительных полей	синтаксис команд чрезвычайно прост, логичен, интуитивно понятен	синтаксис команд довольно сложен, инструкции имеют переменную длину и множество факультативных контекстно-чувствительных полей	синтаксис команд упрощен до предела
система кодировки команд оптимизирована	по компактности	по скорости выполнения и легкости чтения в машинных кодах	неоптимизирован	по скорости выполнения в ущерб компактности
параллелизм	параллелизм не заложен явно, более того, система команд всячески препятствует созданию суперскалярных процессоров	параллелизм не заложен явно, но создание суперскалярных процессоров в данной системе команд осуществляется легко	параллелизм не заложен явно, более того, система команд всячески препятствует созданию суперскалярных процессоров	параллелизм не заложен явно, но система команд оптимизирована под параллельное исполнение
выравнивание	наличие команд длиной в байт, вызывает проблемы с выравниваем кода	все команды кратны размеру слова и потому всегда выровнены	все команды кратны размеру слова и потому всегда выровнены	все команды равны длинному слова и потому всегда выровнены
происхождение набора команд	насильно навязан Data-Point, заказавшей Intel разработку чипа для своих терминалов. стремление руководства Intel обеспечить обратную совместимость процессоров последующих поколений с неизбежностью привела к отказу от лучших решений в пользу уже имеющихся.	оригинальный набор команд, разработанный без учета обратной совместимости, что превратило PDP-11 в могильщика огромного количества ранее написанного программного кода, и причем очень хорошего кода.	базирующийся на PDP-11, но существенно пересмотренный и переработанный набор команд	нет данных (по видимому, оригинальная разработка DEC)
особенности адресации				
минимально адресуемая ячейка памяти	байт	байт	байт	байт
возможность адресации битов	отсутствует	отсутствует	частично реализована	отсутствует?
виды адресации основного процессора	24 вида адресации; примитивная, несимметричная адресация	6 видов адресации, богатая, симметричная	18 видов адресации, богатая симметричная адресация, наследующая все лучшие из PDP-11 и добавляющая к ним все сильнейшие из x86	3 вида адресации, крайне бедная, несимметричная адресация
	регистровая адресация	регистровая адресация	регистровая адресация	регистровая адресация
	непосредственная адресация	непосредственная адресация	непосредственная адресация	–
	косвенная адресация по непосредственному значению, регистру, сумме одного/двух регистров и непосредственного значения с поддержкой масштабирования в 2, 4 и 8	косвенная адресация по непосредственному значению, регистру или сумме регистра с числом	косвенная адресация по непосредственному значению, регистру или сумме регистра с числом с масштабированием на 2, 4 и 8.	косвенная адресация по сумме регистра с 16-битным смещением
	–	дважды косвенная адресация (операнд указатель на указатель)	дважды косвенная адресация (операнд указатель на указатель)	–
	–	адресация с автоувеличением (автоуменьшением) операнда до/после взятия его значения	адресация с автоувеличением (автоуменьшением) операнда до/после взятия его значения	–
виды адресации сопроцессора	регистры сопроцессора объединены в кольцевой стек, адресуемый относительно его вершины, причем ряд команд оперирует только значениями лежащими на вершине стека. адресация команд, взаимодействующих с оперативной памятью –	–	полноценная адресация сопроцессора, включая групповые пересылки данных	вещественные команды используют те же самые способы адресации, что целочисленные

	базово индексная. MMX-команды поддерживают два вида адресации: регистровую и косвенную			
порты адресуются как память	нет	да	да?	да
объем непосредственно адресуемой памяти	4 Гб (12 Гб если использовать раздельные стеки для кода, стека и данных, и 16[20] Гб, если динамическую память вынести в отдельный сегмент	64 Кб	4 Гб	от 2^{43} до 2^{64} в зависимости от реализации
модель памяти	сегментная, линейная	плоская, страничная	плоская	плоская
представление адресов	16/32 бит указатель [+ сегмент]	16 бит указатель [+ базовый адрес страницы]	32 бит указатель	43 или 48 бит, расширяемые до 64 бит и транслируемые в 44-битный физический адрес, указатель
представление смещений (в базово-индексной адресации)	16/32 бит	8/16 бит	8/32 бит	16 бит (позор джунглям!)
возможность получения эффективного адреса	да	нет	да	да
масштабирование	есть, на 2, 4, 8 в любой команде	нет	есть на 2, 4, 8 в любой команде	на 4 и 8 в команде сложения только
поддержка виртуальной памяти	да, встроенная	частично	да, встроенная	да, встроенная
регистры				
типы регистров	целочисленные регистры общего назначения (данных и адресов)	целочисленные регистры общего назначения (данных и адресов)	целочисленные регистры данных	целочисленные регистры общего назначения (данных и адресов)
	–		регистры адресов	–
	–		вещественные регистры сопроцессора.80 бит	вещественные регистры сопроцессора
	кольцевой стек регистров сопроцессора			
	векторные регистры сопроцессора			
	регистры специального назначения	регистры специального назначения	регистры специального назначения	регистр указатель команд
	системные регистры	–	системные регистры	системные регистры
разрядность целочисленных регистров	8, 16, 32	16	32	64
разрядность вещественных регистров	64, 80 и 128 бит	–	80 бит	64?
регистры равноправны	регистры неравноправны: многие команды работают с фиксированными регистрами или огранивают выбор регистра	регистры полностью равноправны	регистры неравноправны и делятся на регистры данных (целочисленные и вещественные) и адресные регистры, но внутри "своих" категорий все они равноправны	регистры равноправны, за исключением того, что вещественные регистры не могут быть указателями
кол-во регистров общего назначения	7 32-битных РОН, к 4 из которых устроены так: L8,H8,L16,32 т.к. трактуются либо как два 8 битных регистра, либо один 16 битный, либо один 32-битный;	6 16-битных регистров	16 РОН – 8 регистров данных и 8 адресных регистров	32 целочисленных регистров
	8 80-битных FPU регистров, начиная с P-III: 8 128-битных XMM регистров	–	8 80-битных регистров сопроцессора	32 вещественных регистра
работа с половинками регистров	частично	нет	нет	нет
взаимодействие со специальными регистрами				
непосредственная адресация регистра указателя стека	поддерживается	поддерживается	поддерживается	–
непосредственная адресация регистра указателя команд	отсутствует	поддерживается	поддерживается	отсутствует
операнды				
размер операндов основного процессора	16, 32 и частично 8 бит	16 бит, ограничено 8 бит	8, 16, 32 и очень ограничено 64 бита	64 бит, для операндов в 8, 16 и 32 доступны лишь операции расширения, чтения/записи
размер операндов сопроцессора	32, 64, 80 и, начиная с P-III, – 128 бит	–	1 бит – 256 байт	32, 64 бит, что позор
операции над операндами памяти	все возможные, при условии, что второй операнд не находится в памяти	все возможные	все возможные	чтение и запись только
операции над портами	чтение и запись (включая циклическую обработку) только, причем, в строго определенные регистры	все возможные	все возможные	чтение и запись только
черная дыра, так же называемая битовой корзиной	отсутствует	отсутствует	отсутствует	присутствует – это регистр R31/F31. чтение дает ноль, запись игнорируется

арифметика				
арифметика	с насыщением (saturation) беззнаковая,	–	–	–
	с насыщением знаковая	–	–	–
	циклическая (wraparound) беззнаковая	циклическая беззнаковая	циклическая беззнаковая	циклическая беззнаковая
	циклическая знаковая	частично: циклическая знаковая	циклическая знаковая	циклическая знаковая
поддержка векторных операций	да, начиная с Pentium MMX	отсутствует	отсутствует	отсутствует
				типы данных
типы данных основного процессора	ограничено биты	?	биты	–
	ограничено битовые поля	?	битовые поля	–
	BCD	–	BCD	–
	строки	–		–
	байтовые целые	байтовые целые	байтовые целые	ограничено байты
	словные целые	словные целые	словные целые	ограничено слова
	двухсловные целые	–	двухсловные целые	ограничено двойные слова
	–	–	четвертные целые	четвертные слова
типы данных сопроцессора	float	–	float	VAX F_floating (32-bit)/ IEEE single (32-bit)
	double		double	VAX G_floating (64-bit) - IEEE double (64-bit)
	extend		extend	–
	16-бит целые		16-бит целые	–
	32-бита целые		32-бита целые	–
	64-бита целые		64-бита целые	–
	64-битные BCD		??-битные BCD	–
	packed byte		–	–
	packed word		–	–
	packed doubleword		–	–
	quadword		–	–
управление ходом выполнения программы				
команды условных/безусловных переходов	присутствуют	присутствуют	присутствуют	присутствуют
условные команды пересылки и/или назначения данных	присутствуют	присутствуют	присутствуют	присутствуют
кол-во и коды условий	C (перенос/заем)	C	C	C (перенос/заем)
	P (четность)	–	–	–
	A (вспомогательный перенос)	–	–	–
	Z (нуль)	Z	Z	Z (нуль),
	S (знак),	N (знак),	N (знак),	–
	O (переполнение)	V (переполнение),	V (переполнение),	–
	–	–	X (расширение)	–
ветвления по условиям	C == 0	C == 0	C == 0	C == 0
	C == 1	C == 1	C == 1	C == 1
	Z == 0	Z == 0	Z == 0	Z == 0
	Z == 1	Z == 1	Z == 1	Z == 1
	S == 1	N == 1	N == 1	–
	S == 0	N == 0	N == 0	
	S == O	S == O	S == O	–
	S != O	S != O	S != O	
	O == 0	V == 0	V == 0	–
	O == 1	V == 1	V == 1	
	P == 0	–	–	–
	P == 1	–	–	–
	C == 1 | Z == 1	C == 1 | Z == 1	C == 1 | Z == 1	C == 1 | Z == 1
	C == 0 & Z == 0	C == 0 & Z == 0	C == 0 & Z == 0	C == 0 & Z == 0
	Z == 0 & S == O	Z == 0 & S == O	Z == 0 & S == O	–
	Z == 1 | S != O	Z == 1 | S != O	Z == 1 | S != O	–
	Z == 1 | S != O	Z == 1 | S != O	Z == 1 | S != O	
	CX == 0	–		–
	ECX == 0	–		–
	–	–		Low Bit Is Clear
	–	–		Low Bit Is Set
влияние флагов сопроцессора на команды управления программой	непосредственно не влияют, поэтому, флаг сопроцессора приходится проталкивать через память в регистр флагов или анализировать его вручную	–	для анализа состояния имеется специальный набор команд	управление прозрачно
команды пересылки изменяют флаги	нет	да	да	команды пересылки как таковые отсутствуют. имеются команды чтения/записи данных в/из памяти из/в регистр, не изменяющие флагов. для пересылки регистра A в регистр Б используйте мат.

				операции, например, ADD Б, А, R31
дистанция условных переходов	4 Гб	64 К	4 Гб	4 Мб (позор)
управление механизмом предсказаний ветвлений	отсутствует, процессор предсказывает ветвления автоматически	отсутствует, процессор не предсказывает ветвления	отсутствует, процессор не предсказывает ветвления	возможность задания направления срабатывания в каждом ветвлении
сохранение старого значение указателя команд при переходе	только при вызове подпрограмме на вершине стека	отсутствует	отсутствует	при любом ветвлении в любой целочисленный РОН
				выравнивание
необходимость выравнивания кода	не требуется	обязательно	???	обязательно
необходимость выравнивания данных	не требуется	не требуется	не требуется	не требуется
				средства автоматического контроля достоверности полученных результатов
прерывание при переполнении	нет	нет	нет?	при вещественной и целочисленной арифметике
прерывание при делении на ноль	только при целочисленной арифметике	?	только при целочисленной арифметике?	только при вещественном делении
комплексная микропрограммная поддержка				
				стековые операции
поддержка стека	один стек	много	много	отсутствует
поддержка очередей	отсутствует	да	да	отсутствует
				процедурные средства
команды вызова процедур	частично – адрес возврата всегда заносится на вершину стека, параметры передаются вручную	да?	адрес возврата может быть сохранен где угодно, аргументы могут передаваться как вручную так и автоматически	отсутствуют
манипуляции с кадром стека	да	нет	да	нет
возврат с автоматическим закрытием кадра стека	есть	есть?	есть	нет
				команды пересылки
пересылка групп регистров	ограничено (только в стек)	нет	да	нет?
пересылка данных периферийным устройствам	да	да	да	нет?
пересылка непосредственных данных	да	да	да	нет
				команды обмена
обмен регистров	да	нет	да	нет?
обмен ячеек памяти	нет	?	да	нет
				циклы
поддержка циклов	весьма ограниченная, – поддерживается лишь цикл, стремящийся к нулю, причем счетчик цикла жестко привязан к регистру ECX/CX, причем команда цикла исполняется крайне неэффективно и ее использование не рекомендуется	поддерживается цикл, стремящийся к нулю, счетчик которого может находится в любом регистре или ячейке памяти	отсутствуют	отсутствуют
				байтовые операции
байтовые операции	расширение извлечение 0 и 1 байта из некоторых регистров, (в MMX: …)	перестановка байтов		по парное сравнение, извлечение, вставка (!), маскирование, заполнение
разбить длинное слово, хранящееся в регистры на байты	нет	нет	да	нет
				битовые операции
битовые операции				
подсчет кол-ва битов и др. битовые команды	нет	нет	нет?	есть
				часто используемые математические операции
команда очистки	отсутствует, используется команда пересылки непосред. нуля или XOR	имеется	имеется	отсутствует, но может читаться "черная" дыра
команда обращения знака	нет	есть	есть	нет?
прочие архитектурные особенности				
встроенные средства отладки	есть, начиная с Pentium чрезвычайно богатые	зачаточные	зачаточные	зачаточные?
мониторинг производительности	есть, расширенный	отсутствует	отсутствует	есть: счетчик циклов
команды предвыборки	"ручная" начиная с P-III и Athlon и автоматическая начиная с P-4	отсутствует	отсутствует	присутствуют
возможность заливки собственных микропрограмм	отсутствует	отсутствует	отсутствует	имеется
выводы				

субъективные впечатления от удобства программирования на ассемблере	очень развитая система команд, работать с ним легко, приятно и удобно за исключением незначительных заморочек с отсутствием адресации память-память и жесткой привязки к регистрам в командах IN, OUT, MUL и DIV	очень элегантный и чрезвычайно удобный в работе ассемблер, "делающий" x86 уже за счет развитой системы адресации.	ликвидирует слабые места PDP и обладает практически всеми приятностями x86	крайне обедненный ассемблер, ручная работа на котором превращается из удовольствия в рутину, тем не менее в нем есть свое очарование, за которого его можно полюбить…
богатство сопроцессора	чрезвычайно богатый набор команд, включая всю тригонометрию и еще много чего	–	богатый набор команд	минимальный набор команд
полнота использования процессора компиляторами	ни один компилятор не реализует возможности x86 процессоров в полной мере (особенно это касается векторных операций), поэтому его программирование на ассемблере вполне оправдано	"язык" PDP-11 процессоров легко отображается на язык Си и потому хорошие компиляторы достаточно полно используют возможности процессора	"язык" 68k процессоров легко отображается на язык Си и потому хорошие компиляторы достаточно полно используют возможности процессора	язык DEC Alpha столь примитивен, что и компилятор, и человек используют его практически с одной и той же эффективностью

>>> врезка из глубины веков…

"The irony is that the original instruction set was theirs, and the original motivation was theirs," Mazor said.

"*…по иронии судьбы, этот основополагающий набор инструкций был не наш, как и исходная постановка задачи*"

Стенли Мазор

Ругая Intel за уродливый (с их точки зрения!) набор инструкций, ее противники зачастую даже и не подозревают, что роль компании Intel в становлении x86 набора инструкций более чем скромна. Собственно, "зачастую" еще слабо сказано. Навряд ли хоть один из спорщиков об этом вообще знает! Но, –все по порядку…

…давным-давно, когда славные шестидесятые уже подходили к концу, а кремниевая революция семидесятых еще и не думала начинаться, молодая и еще совсем неокрепшая компания Intel, неожиданно для себя получила заказ от крупной (по тем временам) японской компании Busicom, поручивший парням из Intel разработку двенадцати микросхем для своих калькуляторов, которым Busicom тогда занималась. Не то, чтобы этот заказ воодушевил руководство Intel, которое прекрасно осознавало, что осуществить такой проект своими силами ей просто нереально, но… в бизнесе главное "застолбить" за собой заказ, а там… авось что ни будь да придумаем! Собственно так и произошло: Ведущий сотрудник Intel примерно в это же самое время обнаружил, что новая метало-оксидно-кремниевая технология дает возможность разместить все компоненты центрального процессора на одной-единственной микросхеме. Сейчас это кажется очевидным, но тогда… Intel просто не поняла, что она изобрела и, проявив чудовищную недальновидность, "слила" эту технологию окончательно замучившей ее Busicom, которая к тому времени уже, вероятно, и не надеялась получить свои микросхемы. В общем, вместо двенадцати заказанных микросхем, Intel предложила сделать всего одну супер-микросхему, реализующую функции всех их. Такой поворот событий не очень-то обрадовал японцев (точнее: совсем не обрадовал). По началу они даже едва не отказались, но, хорошенько подумав, решили, что лучше синица в руке, чем совсем ничего и… согласились.

Приблизительно в это же самое время до руководства Intel доходит что было у них в руках и что они упустили. Все, что остается… "Лететь в Японию и вести с Busicom переговоры", выкупая у ошарашенных японцев только что проданный им чип назад… Впрочем, вернемся к непосредственному предмету нашего разговора – разработке системы команд для вновь созданного микропроцессора.

Система команд первого в мире микропроцессора по-видимому разрабатывалась двумя людьми, – собственно самим Тедом Хофом и присоединившимися к нему впоследствии Стенлом Мэйзором. Другие же источники утверждают, что она была естественным путем содрана с процессоров IBM и Digital. Что ж! Дотошным читателям еще во многом предстоит разобраться, мне же искать описание чипа Intel 4004 и скрупулезно сравнивать его с продукцией остальных компаний просто лень. Как бы там ни было, система команд Intel 4004 не имела будущего, и в скором времени умерла. http://www.cs.cuw.edu/csc/csc425/4004card.html.

История x86 начала свой отсчет несколько позднее. Немногое подробности этой истории уцелели с той поры, – вот практически все, что мне удалось раскопать….

В том же 1969 году Intel получила заказ от техасской компании Computer Terminal Corporation (в последствии переименованной в Data-Point), на интегрирование разработанного ею процессора в кристалл кремния. Обратите внимание, – именно ***интегрирование***, а отнюдь не самостоятельную разработку, как это было в случае с Busicom. Другими словами, от Intel требовалось создать микропроцессор в полном соответствии со всеми предоставленными спецификациями в которые естественно входило и полное описание набора машинных команд, которые должен быть понимать терминал.

Используемая Data-Point система команд была… нет, не плохой, скорее ее можно назвать примитивной, но – для терминалов! – вполне достаточной. Действительно, терминалы (даже интеллектуальные!) не особо-то притязательны и излишняя элегантность и архитектурные изыски им абсолютно ни к чему. Сегодня, когда миллиарды транзисторов стоят дешевле качественных компьютерных корпусов (а не этих дребезжащих китайских жестянок), трудно поверить, что еще совсем недавно за ***каждый*** транзистор приходилось платить и железо с избыточной функциональностью просто не окупалось. Счетные задачи решал центральный процессор, установленный на главной машине. Терминал занимался лишь вводом и отображением данных, т. е. обеспечивал то, что сегодня называется интерфейсом с пользователем. Ну да, разумеется, никакой интерактивности (в сегодняшнем понимании этого слова) там и не ночевало и основным средством обещания с компьютером была командная строка. Соответственно, и набор инструкций был ориентирован на управление, а не на вычисления и математическую обработку. К тому же, поскольку программное обеспечение терминала, зашитое в его ПЗУ не меняется каждый день, на удобство программирования терминального процессора всем было глубоко начхать и никакой элегантности от его системы команд попросту не требовалось (кого из нас, сегодняшних, волнует система команд ну скажем процессоров модема?).

Реализация микропроцессора заняла практически три года. Лучшие инженеры компании, увлеченные своими идеями, работали практически на пределе человеческих возможностей, отдавая процессору и дни, и ночи, и выходные. "*Они являлись на работу к восьми утра и работали до шести вечера, забывая сделать перерыв на обед. Часто они просыпались задолго до восхода солнца , чтобы опробовать идею, внезапно пришедшую им в голову. Часто семьи разваливались*". Когда же "младенец" наконец заработал, компания Data-Point переживала не лучшие времена и потому отказалась выплачивать положенную по контракту сумму, ссылаясь на то, что микропроцессор слишком медлителен и к тому же требует чрезмерно большого количества микросхем поддержки. В качестве отступного компании Intel были переданы все права на систему команд микропроцессора, – не слишком-то хорошее утешение, но все-таки лучше чем совсем ничего.

Рисунок 5 Терминал Data-Point с процессором, построенным на базе мелкой логики

Такой поворот событий признаться сильно озаботил Intel и она начала продвигать микропроцессор 8008 (а именно такое обозначение он получил!) собственными силами. И это – надобно сказать – ей удалось! Новый чип нашел широкое применение не только в микроконтроллерах, но и в первых персональных компьютерах только-только начавших появляться к тому времени. Кстати, одна из распространенных легенд гласит, что первым персональным компьютер был "Альтаир", но это не так! "Альтаир", основанный на 8080 появился несколько позднее. Его опередило достаточно больше количество микро-ЭВМ, и RGS-08 с "Марком" в частности. Конечно, это были очень и очень простые машины, основанные на 8008, который поддерживал набор из 45 команд и мог непосредственного адресовать аж 16 Килобайт (не улыбайтесь, – по тем временам эта величина была весьма нехилой).

Вот так, собственно, все и завертелось. Следующий микропроцессор, представленный Intel, – 8080 не был, да и не мог быть подлинно революционным. Да, разработчики добавили тридцать новых команд, увеличили непосредственного адресуемое пространство до 64 КБ, кое-что исправили по мелочи, но на принципиальное улучшение архитектуры они не отважились. Да и никому, собственно, принципиально новый процессор был нужен?

На базе 8080 процессора было собрано большое количество микрокомпьютеров, обросших за время своего существования большим количеством программного обеспечения. Операционная система CP/M, текстовой редактор WordStar, база данных dBase… популярность этих программ была столь велика, что их веяния прослеживаются и до сих пор! А потому, всякая мысль об отказе обратной совместимости в последующих моделях процессоров представлялась руководству Intel и экономически, и политически убийственной, но… они все-таки сделали это, благодаря чему все мы сидим под x86, а не решись инженеры Intel в свое время на достаточно рискованный шаг по усовершенствованию архитектуры своего процессора, – его бы уже давно вытеснила Motorola 68K или DEC Alpha.

Новое детище – 8086, (кстати, на момент своего рождения заметно обогнавшее и время, и рынок) не обеспечивало с 8080 прямой бинарной совместимости (т. е. не могло выполнять уже откомпилированные программы), и было совместимо лишь на уровне ассемблера, т. е. при наличии исходных текстов проблема переноса решалось простой перекомпиляцией. Система команд претерпела значительные изменения (не все из которых, впрочем, пошли ей на пользу), но вместе с дополнительными возможностями приобрела некоторую плебейскую разношерстность и… печально известную сегментную модель памяти. И откуда эти чертовы сегменты только появились, – вздыхали за траханные программисты начала восьмидесятых - конца девяностых. Неужто эти идиоты из Intel не могли придумать как обойтись без них? Что ж,

попробую ответить. Сегменты собственно и появились потому, что 8086 опередил свое время, ухитряясь адресовать аж 1 Мб памяти вместо "положенных" ему 64 Кб. Действительно, при побайтовой адресации памяти, 16-битные указатели могут "бить" лишь в пределах одного 64 Кб блока, а ведь 8086 и был 16-разрядным! Можно, конечно, пойти на хитрость и адресовать память не байтами, а словами (и некоторые процессоры именно так и делают!), – тогда объем непосредственно адресуемой памяти возрастет до 128 Кб, но… во-первых, это все равно не выход, а во-вторых, такое решение идет в разрез с требованием об обратной совместимости. Наконец, сам объем непосредственно адресуемой память – каким бы большим он ни был, – еще ничего не дает! И чтобы им эффективно воспользоваться необходимо иметь как минимум возможность создания перемещаемых программ, – т. е. попросту говоря уметь загружать программу в любое место памяти. Многие микро компьютеры начала восьмидесятых (в том числе и те, что были собраны на базе 8080) выделяли программам фиксированные участки памяти и потому загрузка нескольких программ становилась весьма проблематичной. (В частности, "Stealth"– отладчики тех лет грузили себя в буфер экранной памяти, отъедая несколько нижних строк, естественно код отладчика отображался на терминале в виде бессмысленного мусора, зато такой отладчик не конфликтовал с программами!). К более подробному обсуждению преимуществ и недостатков сегментной модели памяти мы еще вернемся (см. "….."), а пока же заметим лишь то, что сегментная организация памяти намного удобнее страничной адресации, используемой в машинах типа PDP-11 и намного дешевле 32-разрядных указателей, используемых в процессорах типа 68K.