1 Концепция ISA

1.1 Предпосылки к созданию ISA

Идея *обратной совместимости*: сделать так, чтобы все компьютеры имели одинаковый набор команд и различались лишь внутренним устройством.
Разработчики хотели иметь некий промежуточный уровень между *software* и *hardware*, интерфейс для работы с микроархитектурой. Таким образом появилась **АНК**.

1.2 Определение и назначение

Архитектура набора команд (англ. *instruction set architecture, ISA*) – часть архитектуры компьютера, определяющая программируемую часть ядра микропроцессора. Архитектура набора команд служит границей между аппаратурой и программным обеспечением и представляет ту часть системы, которая видна программисту или разработчику компиляторов.

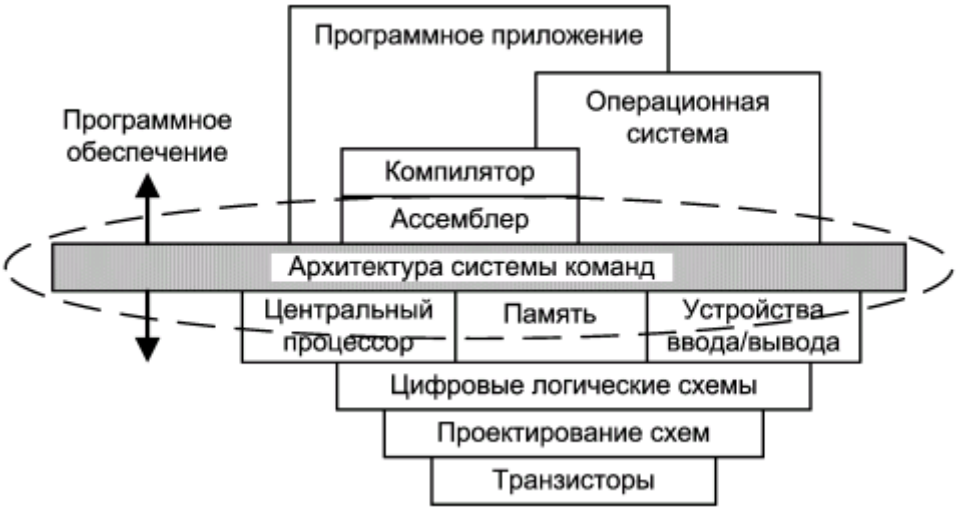


Рис. 1: ISA как интерфейс между АО и ПО

ISA определяет следующие компоненты:

- архитектура памяти (reg-reg, reg-mem и т.д.)
- взаимодействие с внешними устройствами ввода / вывода
- различные типы внутренних данных (целые, знаковые, беззнаковые) и команд ($+$, $-$, $*$, $/$)
- режимы адрессации
- регистры процессора (их количество)
- разрядность адресов
- обработка исключений и прерываний (что происходит при делении на 0, разыменовывание nullptr и т.д.)

Таким образом хорошая ISA должна:

- предлагать набор команд, которые можно эффективно реализовать не только в современной, но и будущей технике
- давать предельную ясность в том, какой именно должна быть откомпилированная программа. Если говорить кратко, поскольку уровень архитектуры набора команд является промежуточным звеном между аппаратным и программным обеспечением, он должен быть приемлемым и для разработчиков аппаратного обеспечения (с точки зрения эффективной реализации), и для программистов (с точки зрения написания качественного кода).

1.2.1 Микроархитектура

В компьютерной инженерии **микроархитектура** (англ. *microarchitecture*), также называемая организация компьютера — это способ, которым данная архитектура набора команд (ISA, АНК) реализована в процессоре. Каждая АНК может быть реализована с помощью различных микроархитектур. К примеру сложение 32-х битного числа может быть реализовано 32-х битным сумматором, а может – с помощью двух 16-ти битных.

Примеры ISA: x86, ARM, MIPS
Примеры микроархитектур: Haswell, Sandy Bridge, Nehalem, Skylake.

2 Классификация архитектур набора команд

2.1 По составу и сложности команд

2.1.1 CISC

CISC (англ. *complex instruction set computer*) – тип ISA, который характеризуется следующим набором свойств:

- нефиксированное значение длины команд
- арифметичкие действия кодируются в одной команде
- небольшое число регистров, каждый из которых выполняет строго определенную функцию
- множество форматов команд различной разрядности
- reg-mem архитектура

Методика построения системы команд CISC противостоит методике, применяемой в другом распространённом типе процессорных архитектур — RISC, где используется упрощённый набор инструкций.
Типичными представителями CISC-архитектуры являются: ранние процессоры на основе команд x86, процессоры мейнфреймов zSeries, процессоры Motorola.

При этом поздние x86-процессоры (Intel Pentium 4, Pentium D, Core, AMD Athlon, Phenom), хотя и CISC-совместимы, но являются процессорами в RISC-ядром, и в формальном смысле считаются гибридными. В таких гибридных CISC-процессорах CISC-инструкции преобразовываются в набор внутренних RISC-команд, при этом одна команда x86 может порождать несколько RISC-команд; исполнение команд происходит на суперскалярном процессоре одновременно по несколько штук. Сделано это с целью освободить кристалл от лишних команд.
Основными недостатками CISC-архитектуры являются:

- высокая стоимость аппаратной части
- сложность с распараллеливанием вычислений
- усложнение аппаратуры вычислительной части, что негативно сказывается на производительности в целом

2.1.2 RISC

RISC (англ. *reduced instruction set computer*) – тип ISA,который характеризуется следующим набором свойств:

- фиксированное значение длины команд (4 байта) с простым форматом
- большое количество регистров общего назначения (32 и более)
- отсутствие микропрограмм внутри самого процессора. То что в CISC выполнялось мирокпрограммами, в RISC иполняется как обыкновенный машинный код
- reg-reg архитектура

Поскольку многие реальные программы тратят большинство своего времени на выполнение простых операций, многие исследователи решили сфокусироваться на том, чтобы сделать эти операции максимально быстрыми. Производительность процессора ограничена временем, которое процессор тратит на выполнение наиболее медленных шагов в процессе обработки любой инструкции; уменьшение длительности таких шагов даёт общее повышение производительности, а также зачастую ускоряет выполнение инструкций за счёт более эффективной конвейеризации. Фокусирование на простых инструкциях и ведёт к архитектуре RISC, цель которой — сделать инструкции настолько простыми, чтобы они легко конвейеризировались и тратили не более одного такта на каждом шаге конвейера на высоких частотах.
Иными архитектурными решениями, типичными для RISC являются:

- Переименование регистров. Каждый регистр процессора на самом деле представляет собой несколько параллельных регистров, хранящих несколько версий значения. Используется для реализации спекулятивного исполнения.

Сравним фиксированную и переменную длину команд:

Фиксированная	Переменная
• Удобнее и проще для декодирования (знаем, что блок из 4 байтов – это 1 команда) • Только короткие команды	• Сложное и медленное декодирования • Можно использовать длинные команды • Частые команды занимают меньше памяти, что критично для кэша

2.1.3 VLIW

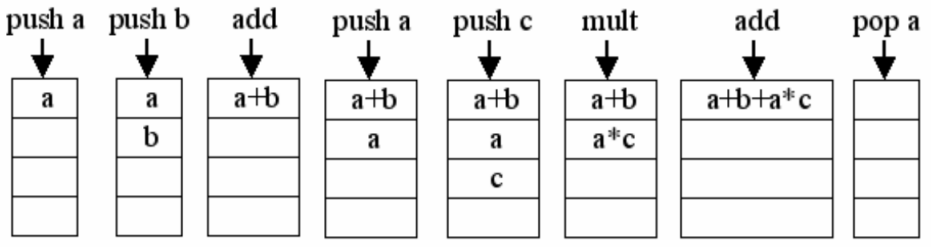
Помимо CISC- и RISC-архитектур в общей классификации также присутствует еще один тип ISA - архитектура с командными словами сверхбольшой длины - **VLIW** (Very Long Instruction Word). Концепция VLIW базируется на RISC-архитектуре, где несколько простых RISC-команд объединяются в одну сверхдлинную команду и выполняются параллельно. В плане ISA архитектура VLIW сравнительно мало отличается от RISC, так как появился лишь дополнительный уровень параллелизма вычислений.

За годы после появления данных архитектур были реализованы и другие альтернативы – MISC, OISC, суперскаляр и так далее.

2.2 По месту хранения операндов

2.2.1 Стековая

Верхнюю ячейку называют вершиной стека. Для работы со стеком предусмотрены две операции: push (проталкивание данных в стек) и pop (вытаскивание данных из стека). Запись возможна только в верхнюю ячейку стека, при этом вся хранящаяся в стеке информация предварительно проталкивается на одну позицию вниз. Чтение допустимо не только из вершины стека. Извлеченная информация удаляется из стека, а оставшееся его содержимое продвигается вверх.
Рассмотрим принцип действия стековой машины на примере $a = a + b + a * c$:



К достоинствам ISA на базе стека следует отнести возможность сокращения адресной части команд, поскольку все операции производятся через вершину стека, то есть адреса операндов и результата в командах арифметической и логической обработки информации указывать не нужно. Код программы получается компактным. Достаточно просто реализуется декодирование команд.
С другой стороны, стековая ISA по определению не предполагает произвольного доступа к памяти, из-за чего компилятору трудно создать эффективный программный код, хотя создание самих компиляторов упрощается. Кроме того, стек становится «узким местом» вычислительных машин в плане повышения производительности.

Одним из примеров, где используется стековая архитектура, является *Java Virtual Machine*.

2.2.2 Аккумуляторная

Возникла одной из первых. В ней для хранения одного из операндов арифметической или логической операции в процессоре имеется выделенный регистр — аккумулятор. В этот же регистр заносится и результат операции. Поскольку адрес одного из операндов предопределен, в командах обработки достаточно явно указать местоположение только второго операнда. Изначально оба операнда хранятся в основной памяти, и до

Рис. 2: Архитектура ЭВМ на на основе аккумуляторной ISA

выполнения операции один из них нужно загрузить в аккумулятор. После выполнения команды результат заносится в аккумулятор, и если этот результат не является операндом для последующей команды, то его требуется сохранить в ячейке памяти.
Рассмотрим принцип работы аккумуляторной архитектуры на примере $z = a + (3b - 2c) * d$:

Команда	Что хранится в аккумуляторе
`Ld 2`	`2`
`Mul [c]`	`2c`
`St [z]`	`z = 2c`
`Ld 3`	`3`
`Mul [b]`	`3b`
`Sub [z]`	`3b - 2c`
`Mul [d]`	`(3b - 2c) * d`
`Add [a]`	`(3b - 2c) * d + a`
`St [z]`	`z = (3b - 2c) * d + a`

Используется в примитивных устройствах.

2.2.3 Регистровая архитектура

Пожалуй, это самый распространенный тип архитектуры. Хотя, конечно, в современных процессорах перечисленные типы архитектуры зачастую объединяются. И так, в

машинах регистрового типа процессор включает в себя массив регистров (регистровый файл), известных как регистры общего назначения (РОН). Эти регистры, в каком-то смысле, можно рассматривать как явно управляемый кэш для хранения недавно использовавшихся данных. Размер регистров обычно фиксирован и совпадает с размером машинного слова. К любому регистру можно обратиться, указав его номер. Количество РОН в архитектурах типа CISC обычно невелико (от 8 до 32), и для представления номера конкретного регистра необходимо не более пяти разрядов, благодаря чему в адресной части команд обработки допустимо одновременно указать номера двух, а зачастую и трех регистров (двух регистров операндов и регистра результата). RISC-архитектура предполагает использование существенно большего числа РОН (до нескольких сотен), однако типичная для таких вычислительных машин длина команды (обычно 32 разряда) позволяет определить в команде до трех регистров. Регистровая архитектура допускает расположение операндов в одной из двух запоминающих сред: основной памяти или регистрах.

С учетом возможного размещения операндов в рамках регистровых ISA выделяют три подвида команд обработки: **reg-reg**; **reg-mem**; **mem-mem**. В варианте reg-reg операнды могут находиться только в регистрах. В них же засылается и результат. Подтип reg-mem предполагает, что один из операндов размещается в регистре, а второй в основной памяти. Результат обычно замещает один из операндов. В командах типа mem-mem оба операнда хранятся в основной памяти. Результат заносится в память. Каждому из вариантов свойственны свои достоинства и недостатки. Ниже приведено сравнение достоинств и недостатков этих операций: (спрошу, нужно ли сравнивать их). Различают **reg-reg-2** и **reg-reg-3**, **reg-mem-2** и **reg-mem-3**, а также **mem-mem-2** и **mem-mem-3** подвиды архитектур. Они характеризуются количеством операндов в записи команды. Приведем пример реализации выражения $z = (a + 2b)/c$ в различных видах архитектур:

reg-reg-2	**reg-reg-3**	**reg-mem-3**
`Ld R1 [a]`	`Ld R1 [a]`	`Mov R1 [a]`
`Ld R2 [b]`	`Ld R2 [b]`	`Add R1 R1 [b]`
`Mul R2 2`	`Mul R2 R2 2`	`Add R1 R1 [b]`
`Add R1 R2`	`Add R1 R1 R2`	`Div R1 R1 [c]`
`Ld R2 [c]`	`Ld R2 [c]`	`Mov [z] R1`
`Div R1 R2`	`Div R1 R1 R2`	
`St [z] R1`	`St [z] R1`	

FAQ

- Что такое ISA и какие функции оно выполняет? Приведите конкретные примеры ISA.
- Что такое обратная совместимость и зачем она нужна?
- Что такое микроархитектура? Приведите конкретные примеры микроархитектур.
- Сравните CISC и RISC архитектуры.
- Как классифицируются архитектуры по месту хранения операндов?
- Назовите особенности стековой архитектуры. Напишите соответствующую реализацию для выражения $z = (4a - 3c - 3) * b + 1$
- Составьте схему работы аккумуляторной архитектуры.
- Сравните различные реализации регистровой архитектуры.
- Реализуйте выражение $z = (a + 4c)/2 + 3b$ с помощью `reg-mem-2` архитектуры.