1 ISA

ISA - *Instruction Set Architecture* - Архитектура Набора Команд.

1.1 Немного истории

Первой ISA была линейка компьютеров IBM под названием System/360. Это была линейка из четырех компьютеров с разной аппаратной частью, соотвественно с разной точностью вычислений, скоростью работы и ценой. Однако, любая программа написанная для простой модели могла запускаться на более сложных моделях линейки без внесения изменений, а также потенциально и любая программа для сложной машины могла выполняться на более простых. Т.е. набор команд для всех машин линейки был одинаков.

1.2 Что такое ISA?

ISA - некоторый документ, который описывает:

- архитектуру памяти,
- набор команд, доступных программисту, пишущему на машинном языке,
- количество регистров,
- доступные типы данных,
- обработку исключений и прерываний (что происходит при делении на 0 и т.д.),
- разрядность адресов,
- протокол взаимодействия с внешними устройствами ввода/вывода,
- режимы адресации,
- разграничение доступа (привелигированный доступ для ОС, пользовательский для пользователя)

1.3 Примеры ISA

x86, x86_64, ARM, MIPS, System/360

2 Архитектуры памяти

2.1 Стековая архитектура

Набор команд by SKKV:

- Арифметика: sub, add, mul, div (для вычитания и деления: первый операнд - то, что первым сняли со стека)
- Доп. арифметика: subr, divr (первый операнд - то, что сняли вторым со стека)

- Операции с памятью: push[a], pop[a]. a - адрес ячейки в RAM.
- Операции со стеком: pop - выкинуть верхнее значение, push StX - добавить на вершину стека копию ячейки с номером X относительно вершины стека (вершина - St0, следующая за ней ячейка - St1 и т.д)

Взаимодействия в целом вида: снять два верхних значения со стека, произвести с ними операцию, положить результат обратно на стек.

2.2 Аккумуляторная архитектура

Можно определить как стек размера 1, или как регистровую архитектуру с одним регистром. Эта единственная ячейка называется аккумулятором. Работает очень медленно, зато просто и дешево производить
Набор команд by SKKV:

- Работа с памятью: LD[a] - загрузить ячейку из памяти, ST[a] - положить ячейку в память. a - адрес ячейки в RAM
- Арифметические операции: mul a, sub a, add a, div a. a - адрес второго операнда в RAM. Первый операнд - значение, лежащее в аккумуляторе. Результат операции помещается в аккумулятор.

2.3 Регистрово-регистровая архитектура, Reg-Reg

Есть фиксированное число регистров R0, R1 ... RN. Итерировать по ним нельзя! Цифры - не индексы, а часть названия.
Есть две вариации Reg-Reg 2 - операторы принимают два операнда.
SUB R1 R2 R1 -= R2
И Reg-Reg 3 - операторы принимают три операнда.
SUB R1 R2 R3 R1 = R2 - R3
SUB R1 R1 R2 R1 -= R2
Набор команд by SKKV:

- Арифметика: SUB, ADD, MUL, DIV. Работают только с регистрами!! Константы допускаются на последнем месте (SUB R1 R1 5 или SUB R1 5).
- Операции с регистрами: MOV R1 R2 - скопировать значение регистра R2 в регистр R1.
- Операции с памятью: LD R1 [a] - загрузить в R1 значение по адресу a. ST [a] R0 - загрузить в ячейку по адресу [a] значение регистра R0.

2.4 Reg-Mem архитектура

Всё так же, как и в Reg-Reg, только один из операндов может быть адресом в памяти (а может и не быть, как напишешь).
В чем преимущество перед Reg-Reg: команды компактнее, следовательно больше команд влезет в кэш.
В чем недостаток: чуть медленнее, больше команд, сложнее реализовать.

2.5 Mem-Mem

Теперь у нас нет регистров, все операнды - ячейки памяти. Ну такое себе, потому что долго.

3 Кодирование команд

Есть два стула:

1. Команды переменной длины.
   Можем часто используемые команды сделать поменьше и выиграть с кэшированием. С другой стороны, такие команды сложнее декодировать.
   В x86 используются команды переменной длины 1..15 байт.

2. Команды фиксированной длины.
   Теперь просто декодировать, но, например, загрузку 32х битной константы в 32х битный регистр придется разбить на две операции.
   В MIPS используются команды фиксированной длины в 4 байта.

4 RISC и CISC

4.1 CISC

CISC - Complex Instruction Set Computer - много разных команд (в VAX, например, было 200-300).
Обычно reg-mem, обычно переменная длина команд.

4.1.1 Преимущества

- быстрее выполняем всякие упопротые сложные команды, т.к. для них есть отдельные аппаратные вычислительные блоки
- лучше с энергоэффективностью

4.1.2 Недостатки

- упоротые команды используются не так уж и часто, а вычислительные блоки занимают место

4.2 RISC

RISC - Reduced Instruction Set Computer - мало команд.
Обычно reg-reg, обычно фиксированная длина команд.

4.2.1 Преимущества

- можем поставить на кристалл больше простых вычислителей и выиграть по скорости за счет параллельности. или можем добавить на кристалл кэша
- быстрее

4.2.2 Недостатки

- энергоэффективность. какое-нибудь программно реализованное шифрование диска быстро сожрало бы аккумулятор, а аппаратным блоком всё ок.

4.3 В итоге

x86 - CISC снаружи, там очень много всяких разных сложных команд, но аппаратно это RISC. Где-то есть небольшой блок, который преобразует CISC команды в RISC команды, которые затем исполняются.
Такой фокус необходим для сохранения обратной совместимости (в ISA можно добавить команд, а вот выкинуть нельзя).

5 Режимы адресации^

Register	ADD R4 R1 R2	regs[R4] <- regs[R1] + regs[R2]
Immediate	ADD R4 R1 5	regs[R4] <- regs[R1] + 5
Displacement	ADD R4 R1 100(R2)	regs[R4] <- regs[R1] + MEM[100 + regs[R2]]
Register Indirect	ADD R4 R1 (R2)	regs[R4] <- regs[R1] + MEM[regs[R2]]
Absolute	ADD R4 R1 [0x123]	regs[R4] <- regs[R1] + MEM[0x123]
Memory Indirect	ADD R4 R1 @(R2)	regs[R4] <- regs[R1] + MEM[MEM[regs[R2]]]
PC Relative	ADD R4 R1 100(PC)	regs[R4] <- regs[R1] + MEM[100 + PC]
Scaled	ADD R4 R1 100(R1)[R5]	regs[R4]<-regs[R1] + MEM[100 + regs[R1] + regs[R5]

- Register - Регистровая адресация. Операнды - данные, хранящиеся в регистре (ну на самом деле не совсем всё так, но упростим)
- Immediate - Непосредственная индексация. Хранение в адресной части самого операнда, а не его адреса. Так можно работать только с константами
- Displacement - Индексная адресация. Обращение к памяти по значению регистра и какому-то константному смещению.
- Register Indirect - Косвенная регистровая адресация - обращаемся к памяти по значению регистра
- Absolute - Абсолютная адресация. Обращение к памяти по константному адресу. В современных архитектурах почти не используется.
- Memory Indirect*** - Косвенная адресация к памяти(?). Обращаемся к ячейке памяти по значению, которое лежит в другой ячейки памяти, адрес которой лежит в регистре. Уххх. Используется, например, в VAX
- PC (IP) Relative* - Адресация относительно счетчика команд.
- Scaled - (?). Сложная адресация: есть смещение, есть значение регистра, а еще есть еще одно значение регистра, умноженное на что-нибудь. Точно есть в x86 (называется SIB - Scaled index base). Удобно, например, если есть массивчик четырехбайтовых слов и хочется по нему удобно перемещаться. Обычно домножать можно не на что угодно, а на степень двойки.

6 Типы данных^

- Binary Integer
- Binary Coded Decimal(BCD) - для более точных вычислений с десятичными дробями (нужно, например, банкам)
- Floating Point - бывает очень разным
  - IEEE 754 - в основном используется
  - Cray Floating Point - была такая штука когда-то, возможно еще живое, юзало нестандартное разбиение на мантиссу и экспоненту
  - Intel Extended Precision (80bit) - у x87 был вычислитель, но чет сейчас обычно Extended типы считаются программно (долго)
- Packed Vector Data. Используется в MMX (всяком мультимедиа). Для таких штук обычно есть специальные SIMD-инструкции, которые весь упакованный массивчик считают за быстро на разных вычислителях. Такое точно поддерживал Pentium MMX
- Address - собственно, адрес. На старых архитектурах бывал отдельный тип под адрес, даже отдельные регистры под адрес

7 Микроархитектура

Микроархитектура - способ реализации конкретной ISA. Микроархитектура определяет такие вещи как: число конвееров, размер кэша, напряжение питания, порядок выполнения команд, ширину шины и тому подобное.
Пример микроархитектуры: MIPS (ISA - MIPS), Intel Core, Pentium (ISA - x86).

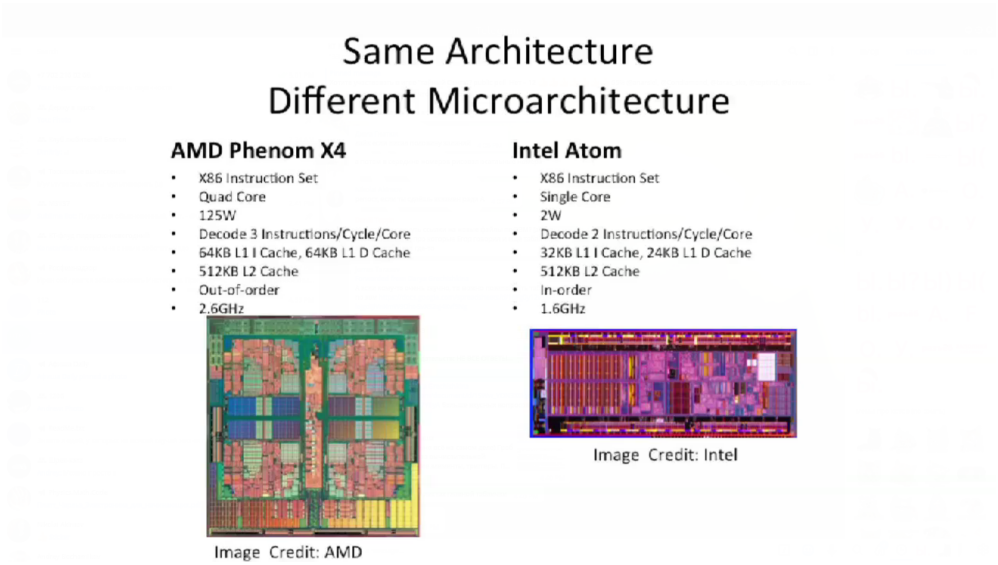

Рис. 1: