Низкоуровневое программирование

Лекция 10

CISC и RISC

Обзор архитектуры ARM

VLIW

CISC-архитектуры

Архитектуры типа CISC (Compete Instruction Set Architectures) исторически появились первыми.

Поскольку на начальном этапе развития ЭВМ многие программы писались на языке ассемблера, набор инструкций CISC-архитектур создавались с учетом удобства программиста.

В частности, в CISC-архитектурах присутствуют «сложные» инструкции, которые могут фактически выполнять несколько действий одновременно (`INC [RAX]` выполнит чтение-сложение-запись).

Наличие сложных инструкций, кодирующих несколько действий, положительно сказывается на размере программы –> экономится память. С целью этой же экономии памяти инструкции в CISC-архитектурах кодируются переменным количеством байт (более редкие инструкции кодируются более длинными последовательностями).

Набор инструкций имеет размер за счет некоторой избыточности – вариаций одной инструкции (`INC [RAX], INC RAX`) и инструкций для специальных случаев (`ADD RAX, 1 ~ INC RAX`).

Оборотной стороной такого подхода является усложнение выборки и декодирования инструкций.

Наиболее известным примером CISC-архитектуры является архитектура x86-64.

RISC-архитектуры

Архитектуры семейства RISC (reduced instruction set computer) имеют уменьшенный набор инструкций, за счет устранения избыточности (есть `ADD R0,1`, но нет `ADD [R0],1` или `INC R0`). В RISC-архитектурах нет инструкций, выполняющих разнотипные действия (вычисления и загрузку/выгрузку).

Существенным плюсом RISC-архитектур является отсутствие необходимости в сложном декодере инструкций. Это упрощает схемотехнику ЦП и увеличивает производительность.

С другой стороны, программе для RISC-архитектуры требуется больше инструкций для достижения того же результата, что (потенциально) ведет к увеличению общего размера программы и снижению эффективности кэширования.

Частично это компенсируется тем, что для RISC-архитектур инструкции кодируются последовательностями одинаковой или кратной длины. Это увеличивает предсказуемость чтения, по сравнению с CISC-архитектурами (для x86-64 инструкция кодируется 1-15 байтами).

Наиболее успешной RISC-архитектурой является ARM.

Advanced RISC Machine

Семейство архитектур набора команд **ARM** является доминирующей в мобильном сегменте. Как следует из названия, ARM – RISC-архитектура.

По аналогии с x86, семейство ARM эволюционировало последовательно. Конкретные архитектуры именуются ARMv*X*, где X – номер. Последней версией является архитектура ARMv9.

Архитектуры ARMv1-7 являются 32-битными. В ARMv8 появился (опциональный для реализации) 64-битный режим работы ЦП - Aarch64.

Регистры в ARM

R0-12 - регистры общего назначения

R13/SP – указатель вершины стека (аналог RSP)

R14/LR – Link Register (LR), хранит адрес возврата из текущей процедуры.

R15 (IP) – программный счетчик (аналог RIP), хранит адрес инструкции, *следующей за следующей*.

PSR/PSTATE – Program Status Register (аналог RFLAGS).

V0-15 – векторные регистры (аналог XMM).

FPCR – управляющий регистр FPU

FPSR – регистр состояния FPU

Основные особенности ARM

1. Load-store архитектура – чтение из памяти выполняется инструкциями `LDR/POP`, запись в память – `STR/PUSH`, остальные инструкции работают только с регистрами (`INC [RAX]` - нельзя).
2. Архитектура-«конструктор» – есть набор обязательных требований и опциональных расширений (например, наличие векторных инструкций, возможность переключения порядка байтов или обязательность выровненного доступа в память).
3. Фиксированная ширина инструкции при стандартной кодировке (32 бита).
4. Специальный режим с укороченной кодировкой (Thumb mode, 16 бит на инструкцию). Переключение осуществляется во время выполнения инструкцией `BX`.
5. Большое количество регистров общего назначения (16 в Aarch32, 32 в Aarch64).

Основные особенности ARM

5. Трехоперандный синтаксис инструкций (`ADD R0, R1, R2` -> R0=R1+R2).
6. Только 32-битные операции – нет деления регистров на части.
7. Наличие LR – при вызове процедуры адрес возврата сохраняется в специальный регистр, что ускоряет вызов листовых функций. Если функция вызывает другие функции, значение из LR должно явно сохраняться на стек инструкцией `PUSH` (https://godbolt.org/z/jshbK1qYz).
8. Арифметические операции имеют по 2 варианта – с обновлением флагов в PSR и без обновления (ADD**S** и ADD).
9. Наличие предикации – многие инструкции могут выполняться или не выполняться в зависимости от результата предыдущего сравнения (например, ADDNE – выполнить сложение ADD при !=). (https://godbolt.org/z/Pavv8GWoE)

Пункты 8 и 9 позволяют писать короткие if-else без условных переходов.

Основные особенности ARM

10. Схема сдвига на входе АЛУ – сдвиг может вычисляться одновременно с другими операциями.

`ADD  R0, R1, R2, LSL #3` -> R0=R1+(R2 << 3)

11. Продвинутые PUSH и POP и инструкции загрузки/выгрузки – вызволяют одновременно сохранять на стек несколько регистров

```
PUSH {R0, R1}            LDM {R1, R2}, [R0]
```

12. Возможность обновления адреса в регистре при чтении/записи

```
LDR R0, [R1, #4]     // R0 = *(R1+4)
LDR R0, [R1, #4]!    // R1+=4, R0=*R1
LDR R0, [R1],#4      // R0=*R1, R1+=4
```

Векторные и вещественные операции

1. По аналогии с SSE, для осуществления скалярных операций используется младшая часть векторного регистра. Например, `D0` и `S0` – 64- и 32-битная части регистра V0.
2. Одни и те же мнемоники используются для векторных и скалярных операций. Ширина вектора указывается вместе с регистром. Например:

```
FADD S0   ,S0    , S1      //скалярная инструкция
FADD V0.2S, V0.2S, V1.2S   //векторная инструкция  (2 float)
FADD V0.4S, V0.4S, V1.4S   //векторная инструкция  (4 float)
ADD  V0.8B, V0.8B, V1.8B   //векторная инструкция  (8 char)
```

Контроль привилегий в ARM

Архитектура ARM определяет 2 уровня привилегий* PL0-1 (соответствуют кольцу 3 и кольцу 0) и 7 режимов работы - User, System, Supervisor, IRQ, Fast IRQ, Undef и Abort. В режиме User применяется PL0, в остальных режимах - PL1.

Режимы User и System имеют полностью общий набор регистров. Остальные режимы имеют собственные регистры SP, LR и SPSR (как следствие – у каждого режима свой стек).

При переключении между режимами содержимое регистра PSR копируется в специальный регистр SPSR (saved PSR), а точка возврата – в регистр LR целевого режима. При обратном переключении состояние PSR и LR восстанавливается.

Переход в режимы IRQ и Fast IRQ происходит автоматически при прерывании (в режиме Fast IRQ обрабатываются приоритетные прерывания).

Системные вызовы (инструкция `SWC`) обрабатываются в режиме Supervisor.

* при поддержке аппаратной виртуализации – +1 уровень привилегий PL2 и режим работы Hyp.

Регистры в Aarch64

X0-29 - регистры общего назначения

X30/LR – Link Register, хранит адрес возврата из текущей процедуры.

X31/ZR - Zero Register, чтение всегда дает 0, запись не изменяет значение

SP – указатель вершины стека (аналог RSP)

PC– программный счетчик (аналог RIP), хранит адрес *текущей инструкции*.

PSR –аналог RFLAGS.

V0-31 – векторные регистры (аналог XMM).

FPCR – управляющий регистр FPU

FPSR – регистр состояния FPU

Особенности Aarch64

1. Длина регистров – 64 бита. Весь регистр обозначается Xn, младшая половина – Wn (по аналогии с RAX-EAX).
2. Операции – либо 64-битные, либо 32-битные – в зависимости от регистра.
3. Большинство инструкций с предикацией убраны.
4. Отсутствие прямого доступа к регистру PC.
5. Отдельный Zero Register XZR(WZR). Всегда содержит 0. ZR можно использовать, как приемник – в этом случае записываемое значение просто потеряется.

Контроль привилегий в Aarch64

В Aarch64 процессор может работать в 4 уровнях исключений (exception levels, EL0-3).

Пользовательский код работает в EL0.

Код ядра ОС и обработчики прерываний работают в EL1.

Опционально, могут быть реализованы уровни EL2 и EL3. EL2 предназначен для работы гипервизора, в EL3 – для монитора безопасности.

ARM и выравнивание данных

В ранних версиях ARM (до ARMv6) доступ к данным должен был быть выровненным по границе в 4 байта (за исключением 1- и 2-байтовых значений – для них применяется выравнивание в 1 и 2 байта). Попытка невыровненного доступа приводила к аппаратному исключению.

Начиная с ARMv6 для обычных инструкций чтения/записи, которые затрагивают 1 регистр, ограничения на выравнивание убрано. В Aarch64 ограничения на выравнивание были ослаблены еще сильнее – невыровненный адрес разрешен для инструкций, которые считывают/записывают несколько регистров.

Специальные инструкции чтения/записи (например, инструкции `LDREX/STREX`, по назначению аналогичные префиксу `LOCK`) требуют выровненного доступа к данным.

ARM и порядок операций с памятью

В отличие от x86-64, где большинство операций с памятью строго упорядочены, в ARM нет ограничений на переупорядочивание (за исключением очевидных ограничений для операций с одной областью). Это положительно сказывается на производительности однопоточных программ, но создает проблемы для многопоточных программ.

Для обеспечения корректности выполнения в ARM используются барьеры памяти, которые ограничивают переупорядочивание операций. С точки зрения рядового программиста это означает, что неверное использование `volatile` вместо атомарных типов в C++ может приводить к трудноуловимым багам на ARM.

https://godbolt.org/z/6oPo16qKa

VLIW-архитектуры

Суперскалярные ЦП (неважно, RISC или CISC) могут выполнять несколько инструкций за такт, но выполняют распределение операций по исполнительным устройствам на лету.

VLIW-архитектуры (Very Large Instruction Word) используют иной подход: одна *длинная инструкция* кодирует несколько действий, каждое из которых отправляется в свое исполнительное устройство. Количество и тип исполнительных устройств (а значит, и предельная емкость инструкции) явно прописывается в ISA.

Распределение инструкций по исполнительным устройствам выполняется на этапе компиляции. Поскольку компилятор обладает большей информацией, он потенциально может лучше распределить операции по исполнительным устройствам.

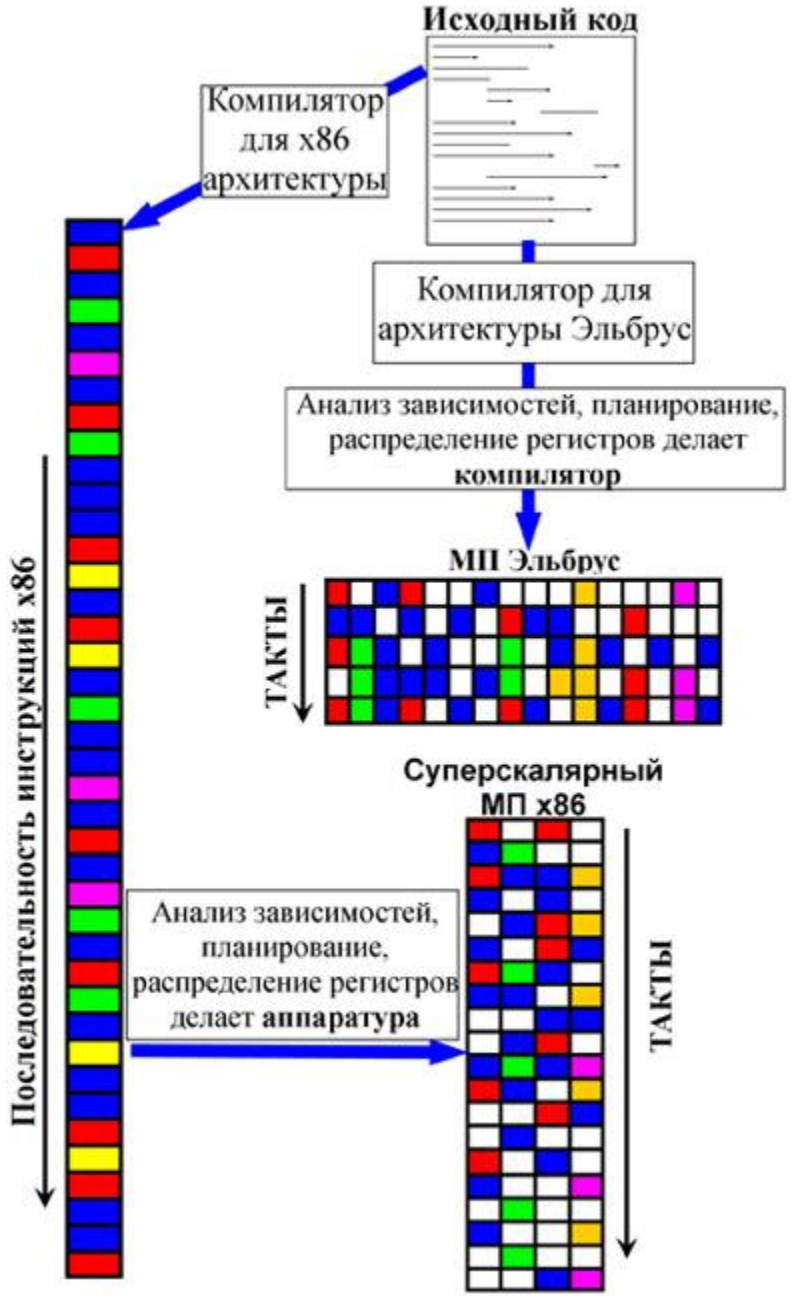
Плюсом VLIW является упрощение ЦП.

Минусом VLIW является зависимость от качества компилятора.

Представители: AMD TeraScale, Qualcomm Hexagon, МЦСТ Эльбрус.

VLIW-архитектуры

https://godbolt.org/z/GdGbefdq1



source