Архитектура ЭВМ ARM-7.

Режимы.

Процессор может находиться в одном из следующих **операционных режимов**:

1. **User mode** — обычный режим выполнения программ. В этом режиме выполняется большинство программ.
2. **Fast Interrupt (FIQ)** — режим быстрого прерывания (меньшее время срабатывания)
3. **Interrupt (IRQ)** — основной режим прерывания.
4. **System mode** — защищённый режим для использования операционной системой.
5. **Abort mode** — режим, в который процессор переходит при возникновении ошибки доступа к памяти (доступ к данным или к инструкции на этапе prefetch конвейера).
6. **Supervisor mode** — привилегированный пользовательский режим.
7. **Undefined mode** — режим, в который процессор входит при попытке выполнить неизвестную ему инструкцию.

Переключение режима процессора происходит при возникновении соответствующего исключения, или же модификацией регистра статуса.

Функции RISC.

Архитектура ARM обладает следующими **особенностями RISC**:

1. Архитектура загрузки/хранения
2. Нет поддержки нелинейного доступа к памяти
3. Равномерный 16x32-битный регистровый файл
4. Фиксированная длина команд (32 бита) для упрощения декодирования за счет снижения плотности кода (позднее режим Thumb повысил плотность кода)
5. Одноцикловое исполнение

Чтобы компенсировать простой дизайнбыли использованы **некоторые особенности дизайна:**

1. Арифметические инструкции заменяют условные коды только когда это необходимо
2. 32-битное многорегистровое циклическое сдвиговое устройство, которое может быть использовано без потерь производительности в большинстве арифметических инструкций и адресных расчетов.
3. Мощные индексированные адресные режимы
4. Регистр ссылок для быстрого вызова функций листьев
5. Простые, но быстрые, с двумя уровнями приоритетов подсистемы прерываний с включенными банками регистров.

Условное исполнение

Одним из существенных отличий архитектуры ARM от других архитектур ЦПУ является так называемая ***предикация*** — возможность условного исполнения команд.

Под ***«условным исполнением»*** здесь понимается то, что команда будет выполнена или проигнорирована в зависимости от текущего состояния флагов состояния процессора.

В то время как для других архитектур таким свойством, как правило, обладают только команды условных переходов, **в архитектуру ARM была заложена возможность условного исполнения практически любой команды**.

Это было достигнуто добавлением в коды их инструкций особого 4-битового поля (**предиката**).

Одно из его значений зарезервировано на то, что инструкция должна быть выполнена безусловно, а остальные кодируют то или иное сочетание условий (флагов).

С одной стороны, с учётом ограниченности общей длины инструкции, это сократило число бит, доступных для кодирования смещения в командах обращения к памяти,

но с другой — позволило избавляться от инструкций ветвления при генерации кода для небольших if-блоков.

Пример:

Алгоритм Евклида.

В языке C он выглядит так:

```
while (i != j)
{
if (i > j)
i -= j;
else
j -= i;
}
```

А на ассемблере ARM — так:

```
loop CMP Ri, Rj   ; set condition «NE» if (i != j),
                  ; "GT" if (i > j),
                  ;or "LT" if (i < j)
SUBGT Ri, Ri, Rj; if "GT" (greater than), i = i-j;
SUBLT Rj, Rj, Ri ; if "LT" (less than), j = j-i;
BNE loop          ; if "NE" (not equal), then loop
```

Из кода видно, что использование предикации позволило полностью избежать ветвления в операторах else и then. Заметим, что если Ri и Rj равны, то ни одна из SUB инструкций не будет выполнена, полностью убирая необходимость в ветке, реализующей проверку while при каждом начале цикла, что могло быть реализовано, например, при помощи инструкции SUBLE (меньше либо равно).

Один из способов, которым уплотнённый (Thumb) код достигает большей экономии объёма — это именно удаление 4-битового предиката из всех инструкций, кроме ветвлений.

Другие особенности

Другая ***особенность набора команд*** – это **возможность соединять сдвиги и вращения в инструкции «обработки информации»** (арифметическую, логическую, движение регистр-регистр).

Например:

Выражение C:

a += (j << 2);

может быть преобразовано в команду из одного слова и одного цикла в ARM:

ADD Ra, Ra, Rj, LSL #2

↓

типичные программы ARM становятся плотнее, чем обычно, с меньшим доступом к памяти.

↓

конвейер используется гораздо более эффективно

ARM процессор также имеет некоторые **особенности, редко встречающиеся в других архитектурах RISC**:

1. адресация относительно счетчика команд (на самом деле счетчик команд ARM является одним из 16 регистров),
2. пре- и пост-инкрементные режимы адресации.
3. некоторые ранние ARM процессоры не имеют команд для хранения 2-байтных чисел.

Конвейер и другие аспекты реализации

ARM7 и более ранние версии имеют **трехступенчатый конвейер:**

1. перенос,
2. декодирования,
3. исполнение.

Cortex-a8 имеет 13-ступенчатый конвейер.

Сопроцессоры

Способ расширения набора команд – **сопроцессоры.**

Могут быть адресованы, используя:

1. MCR,
2. MRC,
3. MRRC,
4. MCRR и похожие команды.

Пространство сопроцессора логически разбито на 16 сопроцессоров с номерами от 0 до 15,

причем 15-й зарезервирован для некоторых типичных функций управления, типа:

1. управления кэш-памятью,
2. операции блока управления памятью.

В машинах на основе ARM **периферийные устройства обычно подсоединяются к процессору**:

1. путем сопоставления их физических регистров в памяти ARM,

или

2. в памяти сопроцессора,

или

3. путем присоединения к шинам, которые в свою очередь подсоединяются к процессору.

Доступ к сопроцессорам имеет большее время ожидания

↓

поэтому некоторые периферийные устройства проектируются для доступа в обоих направлениях.

В остальных случаях разработчики чипов лишь пользуются механизмом интеграции сопроцессора.

Например:

движок обработки изображений должен состоять из малого ядра ARM7TDMI, совмещенного с сопроцессором, который поддерживает примитивные операции по обработке элементарных кодировок HDTV.

Усовершенствованный SIMD (NEON)

Расширение усовершенствованного SIMD, также называемое **технологией NEON** —

это комбинированный 64- и 128-битный набор команд SIMD (single instruction multiple data), который обеспечивает стандартизованное ускорение для медиа приложений и приложений обработки сигнала.

NEON может выполнять декодирование аудио формата mp3NEON может выполнять декодирование аудио формата mp3 на частоте процессора в 10 МГц, и может работать с речевым кодеком GSMNEON может выполнять декодирование аудио формата mp3 на частоте процессора в 10 МГц, и может работать с речевым кодеком GSM AMR (adaptive multi-rate) на частоте более 13МГц.

Он обладает:

1. внушительным набором команд,
2. отдельными регистровыми файлами,
3. независимой системой исполнения на аппаратном уровне.

NEON поддерживает 8-, 16-, 32-, 64-битную информацию целого типа, одинарной точности и с плавающей запятой, и работает в операциях SIMD по

VFP

Технология **VFP (Vector Floating Point, вектора чисел с плавающей запятой)** —
расширение сопроцессора в архитектуре ARM.
Она производит низкозатратные вычисления над числами с плавающей запятой одинарной/двойной точности, в полной мере соответствующие стандарту ANSI/IEEE Std 754—1985 Standard for Binary Floating-Point Arithmetic.

VFP производит вычисления с плавающей запятой, подходящие для широкого спектра приложений:

- КПК,
- смартфонов,
- сжатие звука,
- трёхмерной графики,
- цифрового звука,
- принтеров,
- телеприставок.

Архитектура VFP также поддерживает исполнение коротких векторных команд.
Но, поскольку процессор выполняет операции последовательно над каждым элементом вектора, то VFP нельзя назвать истинным SIMD набором инструкций.
Этот режим может быть полезен в графике и приложениях обработки сигнала, так как он позволяет уменьшить размер кода и выработку команд.
Другие сопроцессоры с плавающей запятой и/или SIMD, находящиеся в ARM процессорах включают в себя FPA, FPE, iwMMXt. Они обеспечивают ту же функциональность, что и VFP, но не совместимы с ним на уровне опкодов.

Расширения безопасности – **TrustZone Technology**,
находятся в ARMv6KZ и других, более поздних, профилированных на приложениях архитектурах.
Оно обеспечивает низкозатратную альтернативу добавлению специального ядра безопасности, обеспечивая 2 виртуальных процессора, поддерживаемых аппаратным контролем доступа.

↓

Позволяет ядру приложения переключаться между двумя состояниями, называемыми **«миры».**Чтобы не допустить утечку информации из более важного мира в менее важный.
Этот переключатель миров обычно ортогонален всем другим возможностям процессора.

↓

каждый мир может работать независимо от других миров, используя одно и то же ядро.

↓

Память и периферия изготавливаются с учетом особенностей мира ядра, и могут использовать это, чтобы получить контроль доступа к секретам и кодам ядра.
Типичные приложения TrustZone Technology должны запускать полноценную операционную систему в менее важном мире, и компактный, специализированный на безопасности, код в более важном мире, позволяя Digital Rights Management'у намного точнее контролировать использование медиа на устройствах на базе ARM, и предотвращая несанкционированный доступ к устройству.

Отладка

Все современные процессоры ARM включают аппаратные средства отладки.

Архитектура ARMv7 определяет базовые средства отладки на архитектурном уровне:

1. точки останова,
2. точки просмотра,
3. выполнение команд в режиме отладки.

Такие средства были также доступны с **модулем отладки EmbeddedICE.**

Поддерживаются оба режима — остановки и обзора.

Реальный транспортный механизм, который используется для доступа к средствам отладки, не специфицирован архитектурно, но реализация, как правило, включает поддержку JTAG.

Существует отдельная архитектура **отладки «с обзором ядра»**, которая не требуется архитектурно процессорами ARMv7.

Поддерживаемые системы ввода-вывода

В большинстве существующих моделей микропроцессоров **реализован**а:

1. шина PCI,
2. возможность работы с внешней динамической оперативной памятью (DRAM).

В процессорах, предназначенных для потребительских устройств, также **обычно интегрируются**:

1. контроллеры шин USB,
2. IIC,
3. AC'97-совместимое звуковое устройство,
4. устройство для работы с флэш-носителями стандарта SD и MMC,
5. контроллер последовательного порта.

Все процессоры имеют линии ввода-вывода общего назначения (GPIO).

В потребительских устройствах к ним **могут быть подключены:**

1. кнопки «быстрого запуска»,
2. сигнальные светодиоды,
3. колесо прокрутки (JogDial),
4. клавиатура.

Регистры

ARM предоставляет 31 регистр общего назначения разрядностью 32 бит.

В зависимости от режима и состояния процессора пользователь имеет доступ только к строго определённому набору регистров.

В ARM state разработчику постоянно доступны 17 регистров:

1. 13 регистров общего назначения (r0..r12).
2. Stack Pointer (r13) — содержит указатель стека выполняемой программы.
3. Link register (r14) — содержит адрес возврата в инструкциях ветвления.
4. Program Counter (r15) — биты [31:1] содержат адрес выполняемой инструкции.
5. Current Program Status Register (CPSR) — содержит флаги, описывающие текущее состояние процессора. Модифицируется при выполнении многих инструкций: логических, арифметических, и др.

Во всех режимах, кроме **User mode** и **System mode**, доступен также **Saved Program Status Register (SPSR).** После возникновения исключения регистр CPSR сохраняется в SPSR. Тем самым фиксируется состояние процессора (режим, состояние; флаги арифметических, логических операций, разрешения прерываний) на момент непосредственно перед прерыванием.

usr	sys	svc	abt	und	irq	fiq
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						R8_fiq
R9						R9_fiq
R10						R10_fiq
R11						R11_fiq
R12						R12_fiq
R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15						
CPSR						
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Форматы команд и способы адресации ARM7.

Набор команд

Чтобы сохранить дизайн чистым, простым и быстрым, оригинальное изготовление ARM было исполнено без микрокода, как и более простой 8-разрядный процессор 6502, используемый в предыдущих микрокомпьютерах от Acorn Computers.

Набор команд ARM

Режим, в котором исполняется 32-битный набор команд.

Набор команд Thumb

В этом режиме процессор выполняет альтернативный набор 16-битных команд.

Большинство из этих 16-разрядных команд переводятся в нормальные команды ARM.

Уменьшение длины команды достигается за счет сокрытия некоторых операндов и ограничения возможностей адресации по сравнению с режимом полного набора команд ARM.

В режиме Thumb меньшие коды операций обладают меньшей функциональностью.

Например:

1. только ветвления могут быть условными,
2. многие коды операций имеют ограничение на доступ только к половине главных регистров процессора.

Более короткие коды операций в целом дают большую плотность кода, хотя некоторые операции требуют дополнительных команд.

В ситуациях, когда порт памяти или ширина шины ограничены 16 битами, более короткие коды операций режима Thumb становятся гораздо производительнее по сравнению с обычным 32-битным ARM кодом, так как меньший программный код придется загружать в процессор при ограниченной пропускной способности памяти.

Аппаратные средства типа Game Boy Advance, как правило, имеют небольшой объем оперативной памяти доступной с полным 32-битным информационным каналом. Но большинство операций выполняется через 16-битный или более узкий информационный канал. В этом случае имеет смысл использовать тумбовый код и вручную оптимизировать некоторые тяжелые участки кода, используя переключение в режим полных 32-битных инструкций ARM.

Первым процессором с декодером тумбовых команд был ARM7TDMI. Все процессоры семейства ARM9, а также XScale, имели встроенный декодер тумбовых команд