For Distribution

Презентация на семинаре Microchip Masters Russia

Юрий Панчул,
24 октября 2013 года

www.imgtec.com

Темы презентации

- Суть бизнеса Imagination Technologies
- История MIPS, подразделения Imagination
- Обзор поколений ядер MIPS – Classic, Aptiv и Warrior
- Ядра MIPS для микроконтроллеров Microchip
- Мини-презентации об избранных деталях MIPS и PIC32
- Сравнения MIPS и ARM
- Рекомендуемая литература

Мини-презентации об избранных деталях MIPS и PIC32

- **Кэши в ядрах MIPS и микроконтроллерах PIC32**
  - Кэш в MIPS microAptiv UP / Microchip PIC32MZ
  - Кэш предварительной выборки в PIC32MX
- **Устройство управления памятью в процессорах MIPS**
  - Буффер ассоциативной трансляции, Translation Lookaside Buffer (TLB)
  - Карты виртуальных адресов в архитектуре MIPS
  - Трансляция с фиксированным отображением, Fixed Mapping Translation (FMT)
- **Матрица шины PIC32MX, Bus Matrix (BMX)**
  - Контроллер прямого доступа PIC32MX, пример использования для графики
- **Соединение PIC32MX с ПЛИС / ППВМ / FPGA**

For Distribution

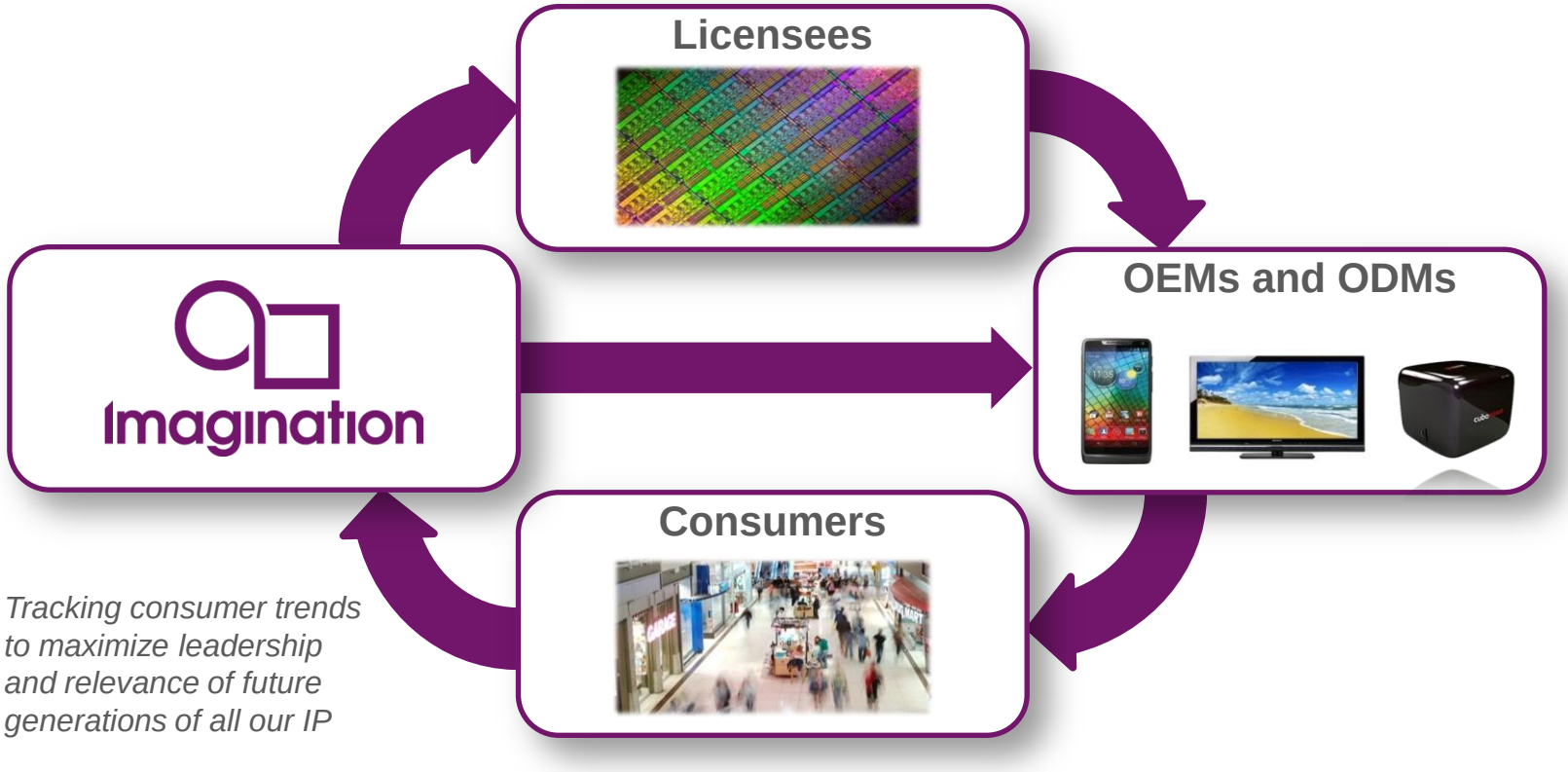
Суть бизнеса Imagination Technologies

www.imgtec.com

Imagination Technologies

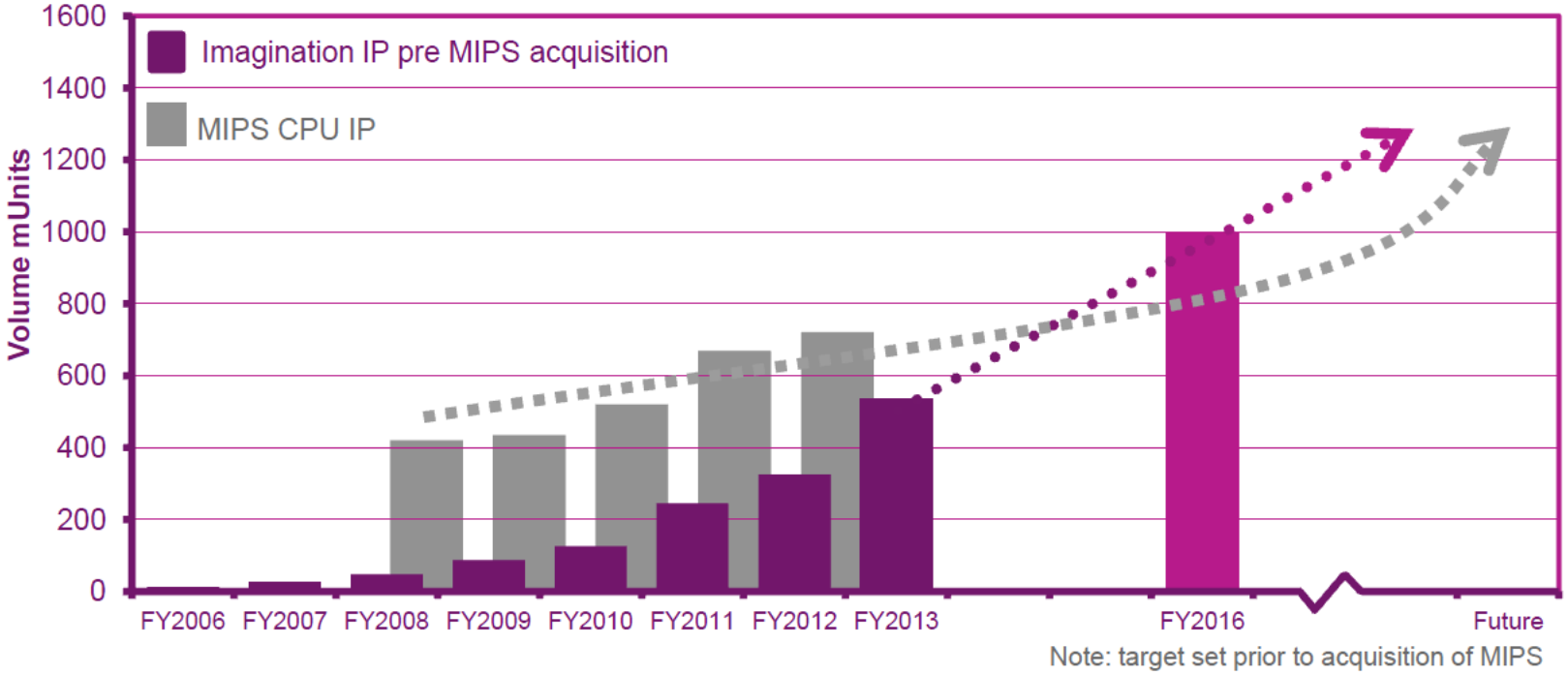
- **Международная технологическая компания**
  - Штаб-квартира в Великобритании
  - Центры разработки в Великобритании, США, Польше, Индии, Китае, Австралии и Новой Зеландии
  - Более 1600 сотрудников
- **Компоненты для систем на кристалле (System on Chip, SoC)**
  - Графический процессор PowerVR GPU
    - Используется в Apple iPhone, iPad и Google Glass
  - Центральный процессор MIPS CPU
  - Видео процессор PowerVR VPU
  - Процессор радио коммуникаций Ensigma RPU
- **В 2013 году частью Imagination стала компания MIPS Technologies, разработчик процессорных ядер MIPS M4K и MIPS microAptiv, которые используются в микроконтроллерах Microchip PIC32**

Бизнес-модель Imagination Technologies



Tracking consumer trends to maximize leadership and relevance of future generations of all our IP

Продажи (в миллионах компонент) MIPS и Imagination



For Distribution

История MIPS, подразделения Imagination

www.imgtec.com

Что такое MIPS?

- **MIPS – одна из популярных RISC архитектур**
  - Возникла в Стенфорде в 1981 году
- **MIPS Technologies – компания, которая в 1999-2012 занималась разработкой ядер с MIPS архитектурой и лицензированием архитектуры MIPS как таковой**
  - Сейчас – подразделение Imagination Technologies
  - Лицензиаты ядер могут встраивать их в свои системы на кристалле – Microchip, Sigma, PMC Sierra
  - Лицензиаты архитектуры могут разрабатывать свою микроархитектуру – Broadcom, Cavium, Академия наук КНР
- **За 2011 финансовый год в мире было выпущено более 656,000,000 устройств с ядром MIPS, за всю историю 3,600,000,000**
  - Процессоры MIPS стоят в цифровых телевизорах Sony, роутерах Cisco, микроконтроллерах Microchip PIC32, фотоаппаратах Samsung и Casio

Вехи истории MIPS – прошлое

- **1981 – начало проекта в Стенфорде**
  - Руководитель проекта Джон Хеннесси сейчас - президент Стенфорда
- **1984 – коммерциализация – MIPS Computer Systems**
- **1991 – первый в индустрии 64-битный микропроцессор – MIPS R4000**
- **1992 – MIPS Computer Systems становится частью Silicon Graphics**
  - Использование в Голливуде и игровых приставках Sony PlayStation и Nintendo64
- **1998 – Компьютерная индустрия напугана анонсированием процессора Intel Itanium; Silicon Graphics решает пустить MIPS в свободное плавание**
  - PC Magazine. How the Itanium Killed the Computer Industry. By John Dvorak. January 26, 2009
  - Itanium так и не состоялся; MIPS продолжил жить во встроенных устройствах

Вехи истории MIPS – современность

- 1998 - MIPS снова становится отдельной компанией и выходит на биржу второй раз (делает второе IPO) как MIPS Technologies

- 1999 – Архитектура MIPS32 и MIPS64
- 2001 – Лицензируемые 32-битные ядра
- 2002 – Ядра со специализацией для микроконтроллеров
- 2005 – Расширение для цифровой обработки сигналов – DSP Extension
- 2006 – Многопоточность на одном ядре – MT (Multi-Threading) Extension
- 2007 – Суперскалярное ядро
- 2008 – Когерентная многопроцессорность
- 2010 – Новая 16-битная система команд microMIPS
- 2012 – Новое поколение ядер – Aptiv Generation

- 2013 – MIPS Technologies становится подразделением Imagination Technologies

For Distribution

Обзор поколений ядер MIPS – Classic, Aptiv и Warrior

Классификация современных процессоров MIPS

- **«Классические» (но не «исторические») ядра от MIPS Technologies**
  - MIPS 4KE, M4K, M14K, M14Kc – малый размер, цена, энергопотребление
  - MIPS 24K, 34K, многоядерный 1004K – эффективность по производительности / милливатт
  - MIPS 74K, многоядерный 1074K – суперскаляр, высшая производительность

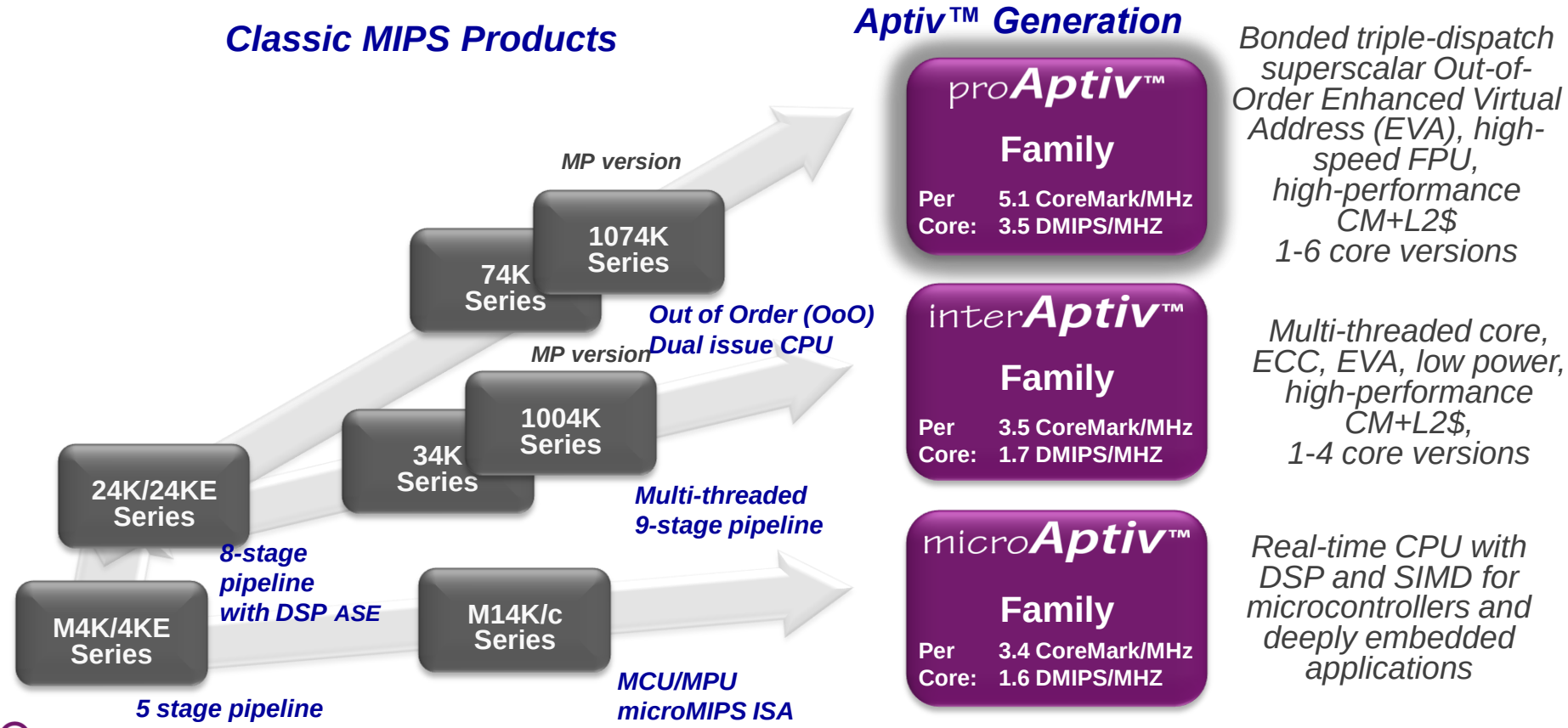
- **Новое (2012) поколение ядер от MIPS Technologies – Aptiv Generation**
  - microAptiv – продолжает линейку M14K, добавляет DSP
  - interAptiv – продолжает 1004K, добавляет улучшенный менеджер когерентности
  - proAptiv – на 60-70% производительнее чем 1074K, дополнительные конвейеры ALU

- **Новейшее (2013 -) поколение ядер от Imagination Technologies – Warrior Generation**
  - Анонсированный MIPS P5600 – основан на proAptiv, добавляет SIMD и виртуализацию

- **Ядра от лицензиатов архитектуры MIPS**
  - Ingenic – 32-битное ядро с 8-стадийным конвейером, очень низкое энергопотребление для своего класса
  - Broadcom/NetLogic, Cavium – высокопроизводительные 64-битные многоядерные сетевые процессоры
  - Loongson – 64-битный суперскалярный процессор от Академии Наук Китая для применений от ноутбуков с Линуксом до серверов и суперкомпьютеров

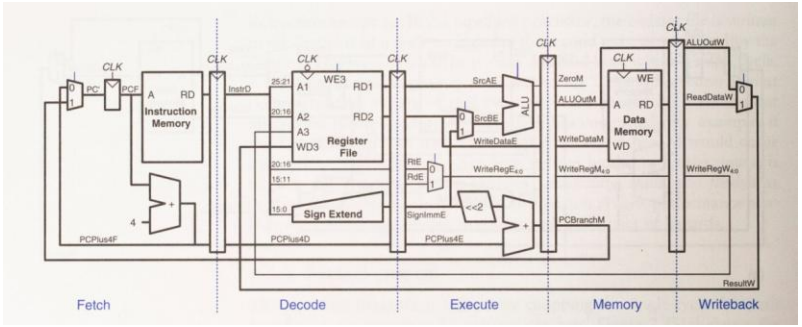
«Классические» ядра и поколение MIPS Aptiv

Flexibility, Scalable Performance at Efficient Power/Cost

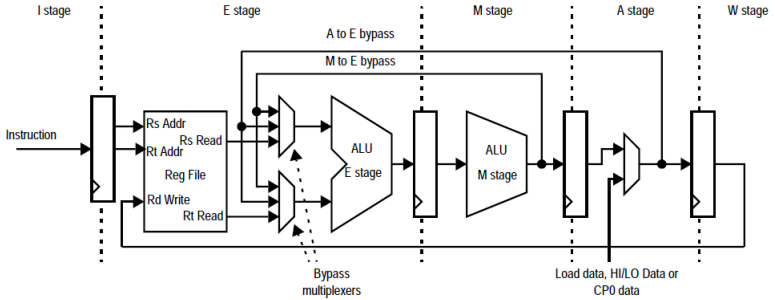


Конвейер M4K напоминает конвейер из учебников

- **Сверху – конвейер процессора, реализующего подмножество архитектуры MIPS из учебника**
  - David Harris and Sarah Harris. Digital Design and Computer Architecture, 2-nd edition. 2012.



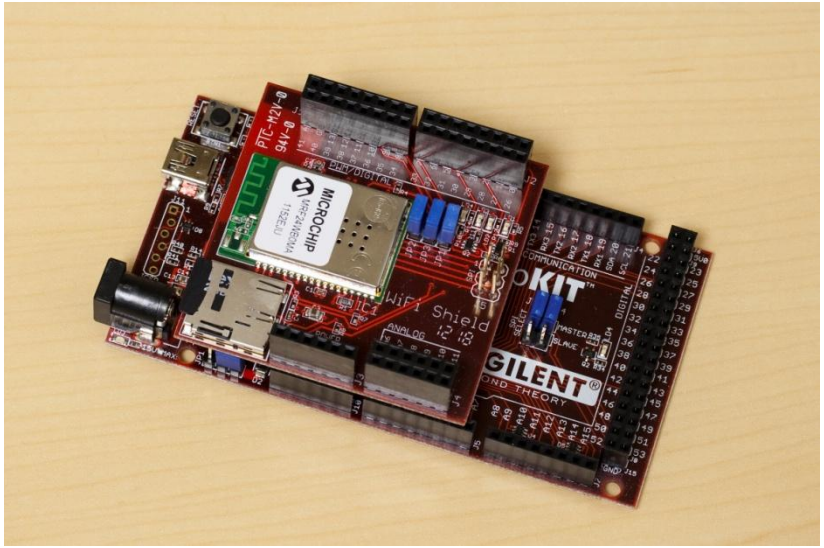
- **Снизу – конвейер индустриального процессора MIPS M4K**
  - MIPS32® M4K™ Processor Core Software User's Manual



Сохраняя преемственность от элегантного академического дизайна, индустриальный MIPS M4K оптимизирован по таймингу и содержит много опций

Демо: RetroBSD на Microchip PIC32

- RetroBSD – версия Unix для микроконтроллеров Microchip PIC32 на основе ядра MIPS M4K
- http://retrobsd.org
- Создана Сергеем Вакуленко – сотрудником MIPS Technologies

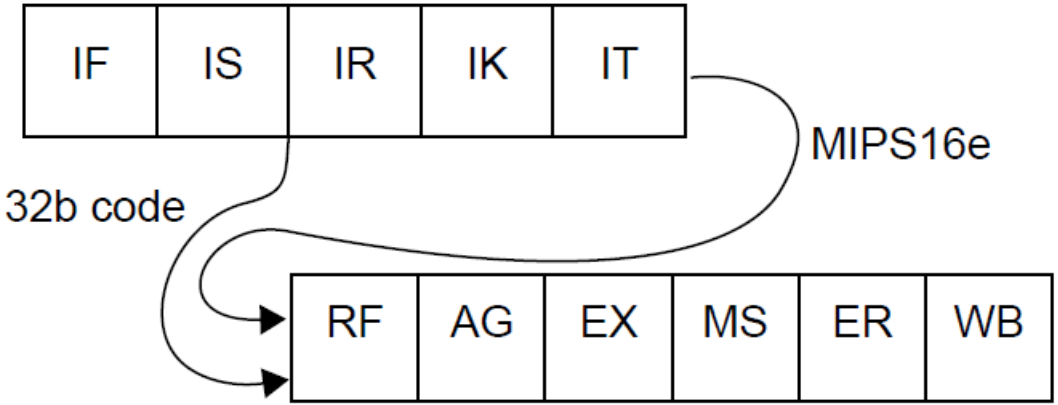


MIPS 24K – история стабильного успеха

- **Эффективное ядро средней производительности**
  - Баланс производительности и размера / энергопотребления
- **8 стадий конвейера (11 стадий в режиме MIPS16e)**
- **Вышло в 2004 году и с тех пор стабильно успешно**
  - Лицензии на использование этого ядра купили более 50 компаний, включая Atheros (куплен Qualcomm), Cisco, Lantiq, Ralink, Toshiba и другие
- **Гибкая поддержка виртуальной памяти с Translation Lookaside Buffer (TLB)**
- **Вариант MIPS 24Kf поддерживает арифметику с плавающей точкой**
- **Вариант MIPS 24KE поддерживает расширение для DSP**
- **1.47 GHz на процессе 40 nm G TSMC, 1.6 DMIPS / MHz, 2.69 Coremark / MHz, 0.10 mW / MHz, 0.36 mm^2**

Конвейер MIPS 24K – 8 стадий

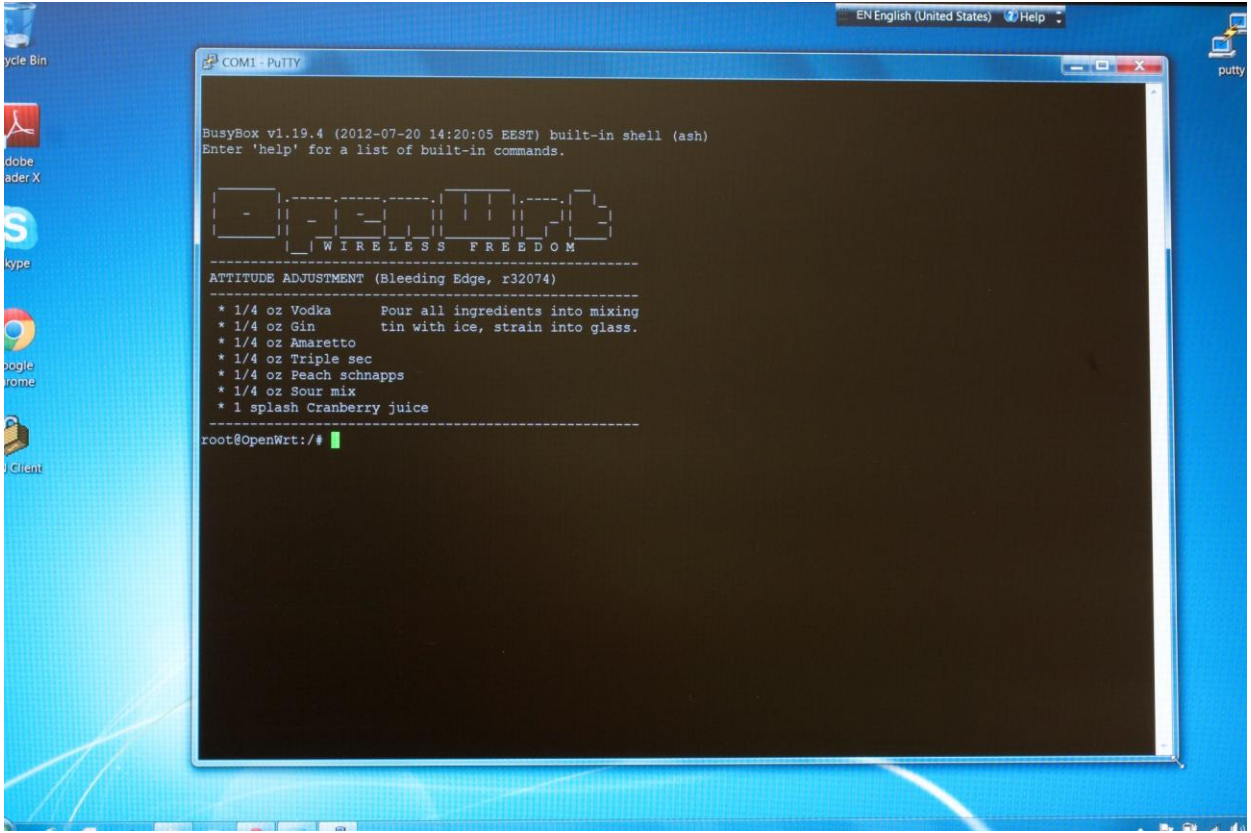
- **Конвейер средней длины**
  - Длиннее, чем у 5-стадийных ядер без предсказателя переходов
  - Короче, чем у суперскалярных ядер с конвейером в 14 стадий и выше
- **Предсказатель переходов очень полезен для 8 стадий**
  - Минимизирует сброс конвейера



Демо: Линуксный компьютер за 22 евро на MIPS 24KE

- Сделан в Литве
- 8devices.com
- Ralink RT3050
- MIPS24KEc
- 320 MHz
- OpenWrt Linux
- На сайте компании есть пример робота и станции наблюдения погоды

Демо: Терминал линуксного компьютера с MIPS 24KE

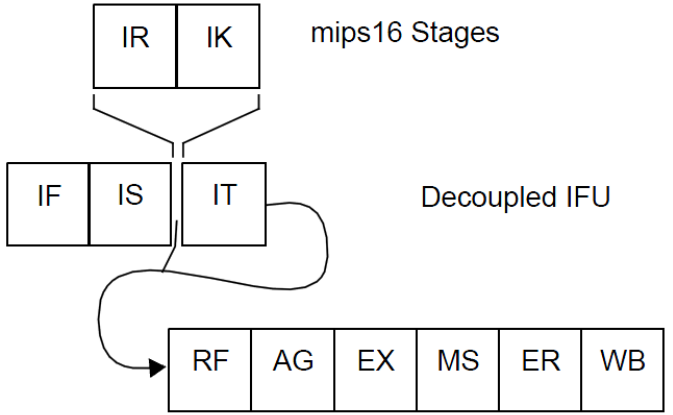


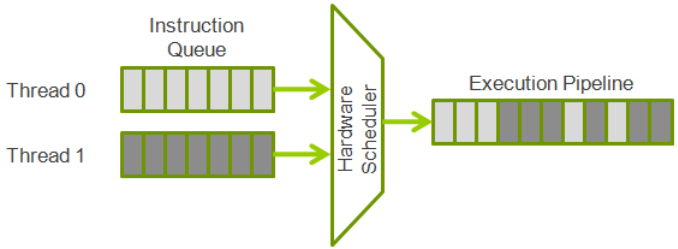
MIPS 34K – многопоточность на одном ядре

- **Ядро MIPS34K основано на MIPS24K с добавлением многопоточности**
  - Выборка иструкций из памяти происходит для нескольких (до 9) потоков (тредов)
  - Пока один тред долго ждет, инструкции из других тредов могут проходить через конвейер процессора
  - Пример ожидания: загрузка данных из памяти, если этих данных нет в кэше, может занимать до 150 циклов и выше
- **Позволяет повысить производительность системы на 20-40% с очень малым увеличением размеров ядра по сравнению с 24K**
  - При этом значительное повышение производительности требует поддержки операционной системы и возникает не на всех задачах
- **1.45 GHz на 40 nm G TSMC, 2.97 Coremark / MHz, 1.6 DMIPS / MHz, 0.46 mm^2, 0.11 mW / MHz**

Конвейер MIPS 34K – 9 стадий

- Дополнительная стадия конвейера позволяет ядру решать, из какого треда выполнить следущую инструкцию
- Решение принимается на основе информации из блока Policy Manager, который может быть модифицирован разработчиком системы на кристалле
- Также разработчик может менять блок Inter-Thread Communication Unit (ITC), который служит для эффективного взаимодействия между тредами

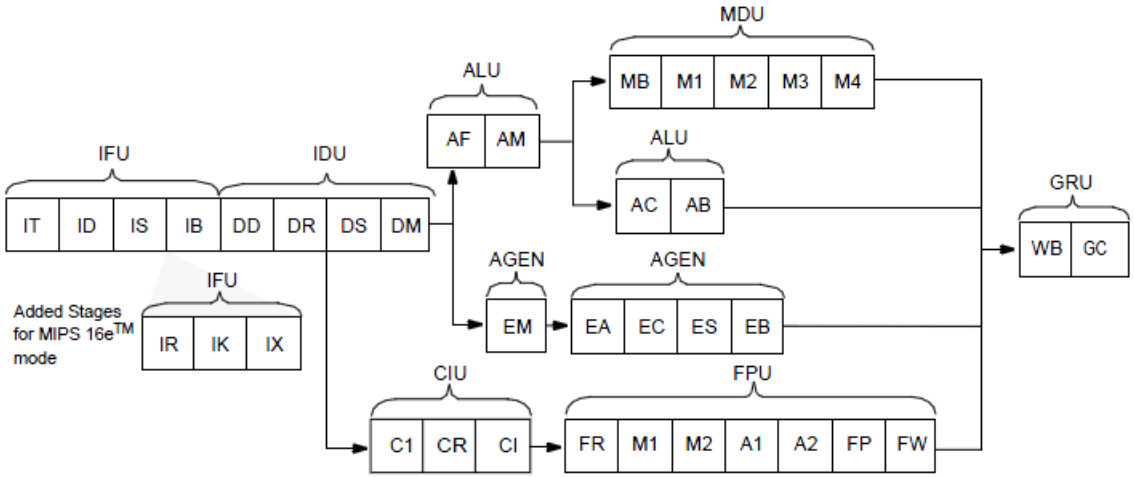




MIPS 74K – суперскалярная производительность

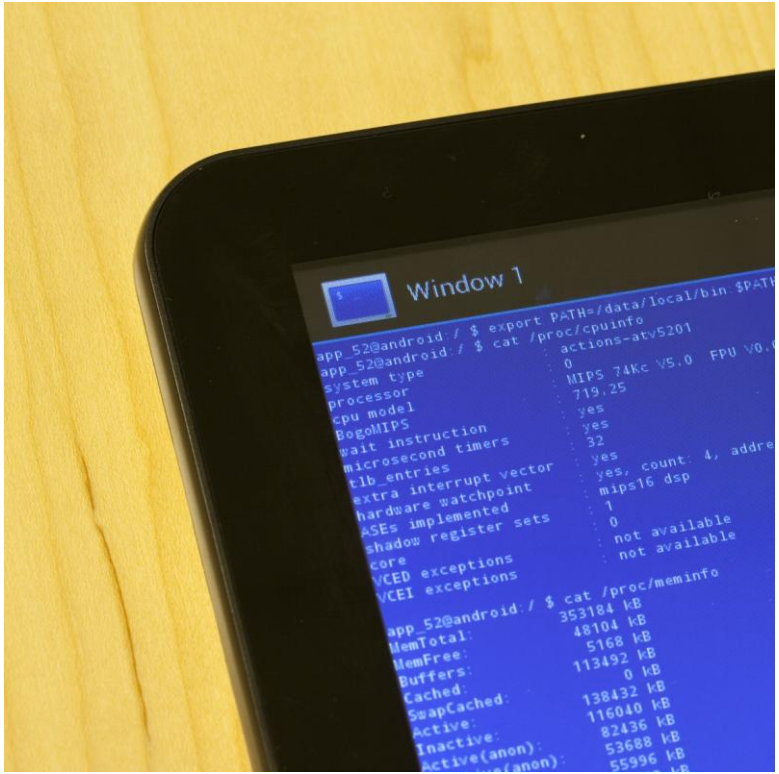
- **Асимметричный конвейер с 15 стадиями, выборкой до 4 инструкций за цикл и out-of-order (OoO) dispatch**

- **1080 MHz на 65 nm GP, 2.57 Coremark / MHz, 2.03 DMIPS / MHz, 0.52 mW / MHz, 1.7 mm^2 без L2$, 2.5 mm2 с L2$**



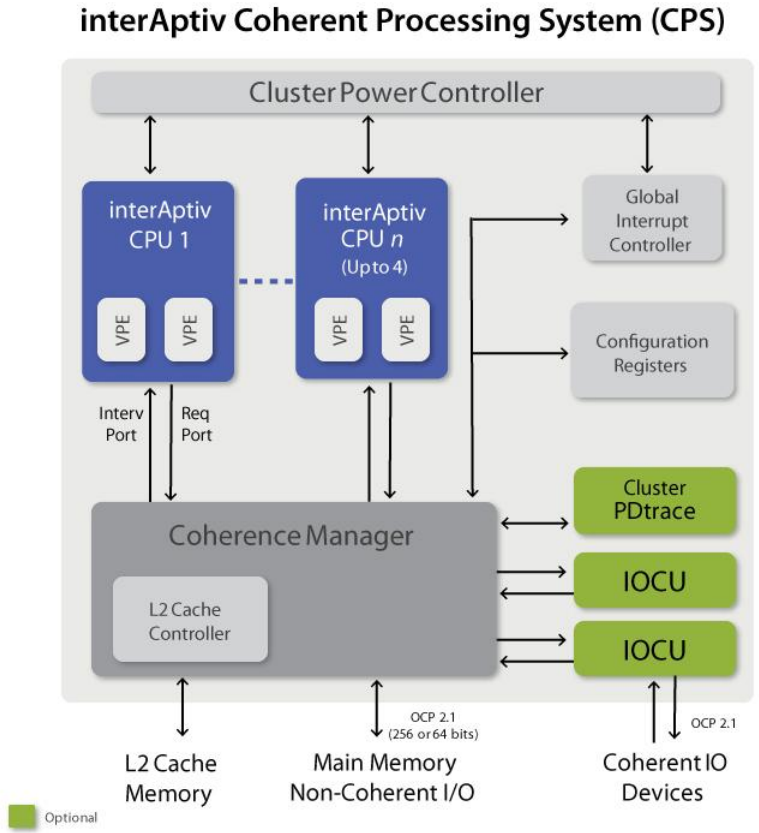
Демо: Андроидный планшет на MIPS 74K

Китайская компания Action Semiconductor лицензировала ядро MIPS 74K и сделала на его основе систему на кристалле для использования в недорогих низкопотребляющих планшетах с Андроидом



Многоядерные системы MIPS 1004K и interAptiv

- Базовые ядра CPU в MIPS 1004K CPS и interAptive CPS основаны на 34K
- Менеджер когерентности позволяет ядрам «подсматривать» в кэши первого уровня друг у друга и не мешать друг другу в работе с общей памятью
- Многоядерная система имеет гибкую систему контроля энергопотребления и другие опции (см. отдельную презентацию)
- Менеджер когерентности для interAptiv (CM2) интегрирован с кэшем второго уровня и имеет оптимизированную latency



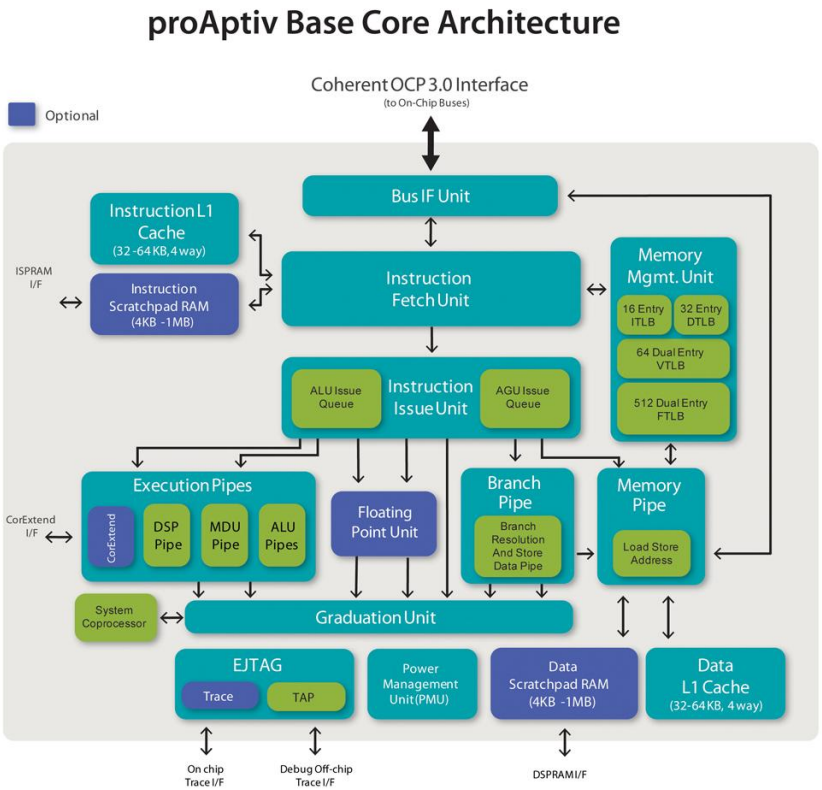
Пример использования MIPS 1004K - Mobileye

- Mobileye – компания с штаб-квартирой в Нидерландах и центром разработки в Израиле
- Mobileye лицензировала MIPS 1004K для своего продукта EyeQ3
- Используется в автомобилях BMW, GM, Volvo and Yulon Motors (Nissan) для предотвращения столкновений на дорогах

Картинка с http://www.gizmodo.com

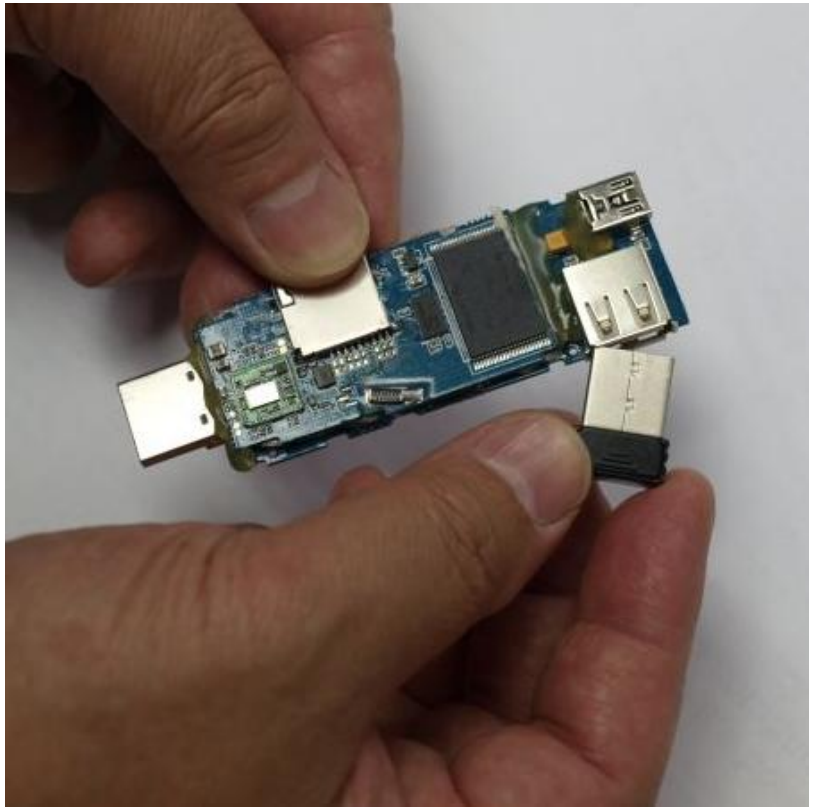
MIPS proAptiv – новый уровень производительности

- **Многоядерный комплекс, как и MIPS 1074K, но:**
  - Базовое ядро proAptiv гораздо производительнее чем 74K за счет дополнительных конвейеров ALU
  - Новый менеджер когерентности (CM2) сильно ускоряет обмен между ядрами и L2 кэшем
  - Имплементирован механизм Enhanced Virtual Address (EVA) для лучшей утилизации адресного пространства



Демо: MIPS-based Ingenic и Андроид в телевизоре

- iPPea TV Dongle – втыкается в телевизор с HDMI
- http://www.ippea.com
- Ingenic Jz4770 1.2 GHz
- Ingenic – лицензиат архитектуры MIPS
- Очень низкопотребляющий процессор
- Android 4.03
- Разрешение 1080p



For Distribution

Ядра MIPS для микроконтроллеров Microchip

www.imgtec.com

Чем ядра MIPS M4K, M14K и microAptiv хороши для микроконтроллеров?

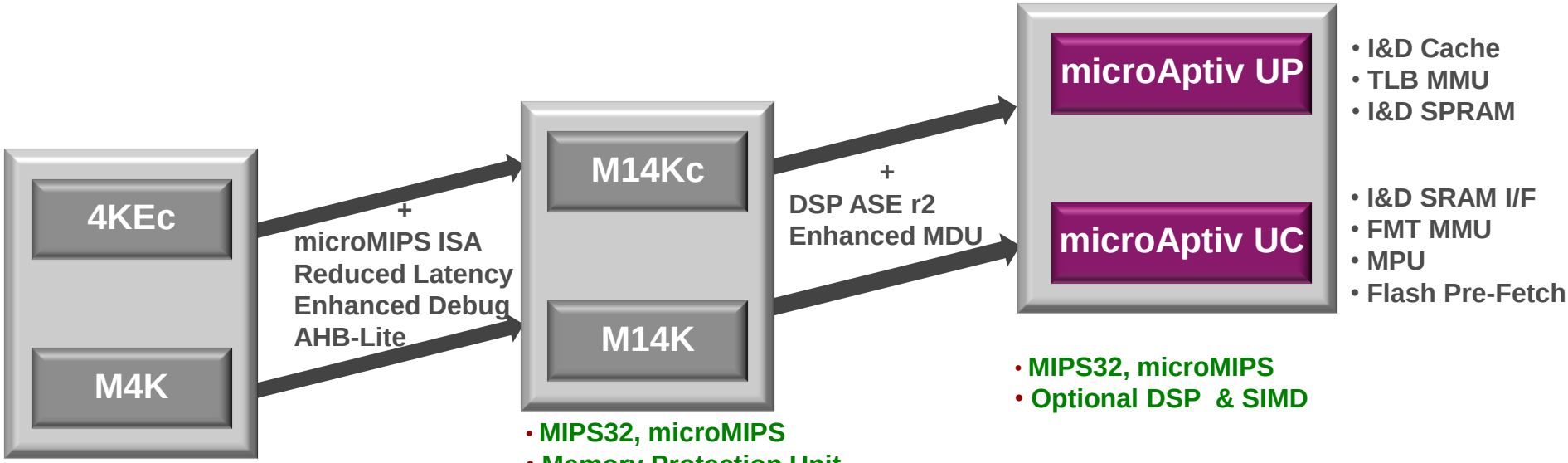
- **Наилучший баланс между производительностью, энергопотреблением и ценой в своем классе**
- **Программная совместимость со всем спектром устройств с архитектурой MIPS**
  - От микроконтроллеров до бытовой электроники и сетевых устройств
- **Зрелые и хорошо оптимизирующие компиляторы**
- **Большое количество RTOS-ов и другого программного обеспечения, написанного для архитектуры MIPS**
- **Возможность использования микроконтроллеров на основе MIPS для целей образования**
  - MIPS широко используется в университетах в курсах по компьютерной архитектуре, дизайну цифровой логики и программированию на языке ассемблера

Лучшая производительность в своем классе

- **MIPS M4K**
  - 1.5 DMIPS / MHz
  - На технологии 90 nm G может работать на 340MHz
  - В Microchip PIC32 работает на частоте 80 MHz
- **MIPS M14K**
  - 1.57 DMIPS / MHz
  - 2.72 CoreMarks / MHz
  - На технологии 65LP может работать на 400 MHz
- **MIPS microAptiv**
  - 1.57 DMIPS / MHz
  - 3.09 CoreMarks / MHz в режиме microMIPS (16-битный набор инструкций)

MIPS32® MCU/MPU Embedded Processor Cores

Same Architecture & ISA
Same Development Tools



- MIPS32, MIPS16e
- 5-stage pipeline
- 1.5 DMIPS/MHz
- Low area & power

- MIPS32, microMIPS
- Memory Protection Unit
- 2-wire Debug
- Up to 30% code size reduction

- MIPS32, microMIPS
- Optional DSP & SIMD

microAptiv DSP Enabled Core Specifications

Target Specs

microAptiv (DSP Enabled)	MCU		MCU		MPU
Process	90LP		65LP		65G
Prod Freq (MHz)	235		380		500
Core Area (mm^2)	0.42		0.24		0.32
Core Active Power (mW/MHz)	0.16		0.08		0.10
Library	9T-SVt		9T-LVt		9T-SVt

Frequency, power consumption and size depend upon configuration options, synthesis, silicon vendor, process and cell libraries

- Production frequency, PTSI, +/- 5% OCV, 100ps clock jitter
- Core Area = **Std cell**
- **MCU = Speed Optimized – microMIPS+MCU ASE+DSP ASE + Fast MDU + Scan+Prefetech+AHB+Memory Protection**
- **MPU = Speed Optimized – microMIPS+MCU ASE+DSP ASE + Fast MDU +Scan+16 TLB MMU + AHB. Memory configuration – 8KB/8KB I/D Cache**

Embedded / MCU

- 150MHz – 90LP
- Real time
- Flash/SRAM
- DSP ASE
- MPU Security
- RTOS/Linux

Mobile

- 300MHz – 65LP
- Real time
- Flash/SDRAM
- DSP ASE
- MPU/MMU
- RTOS/Kernel

Networking

- 400MHz – 65G
- High throughput
- Cache/SDRAM
- DSP ASE
- MPU/MMU
- RTOS/Linux

Strong Industry MCU Adoption

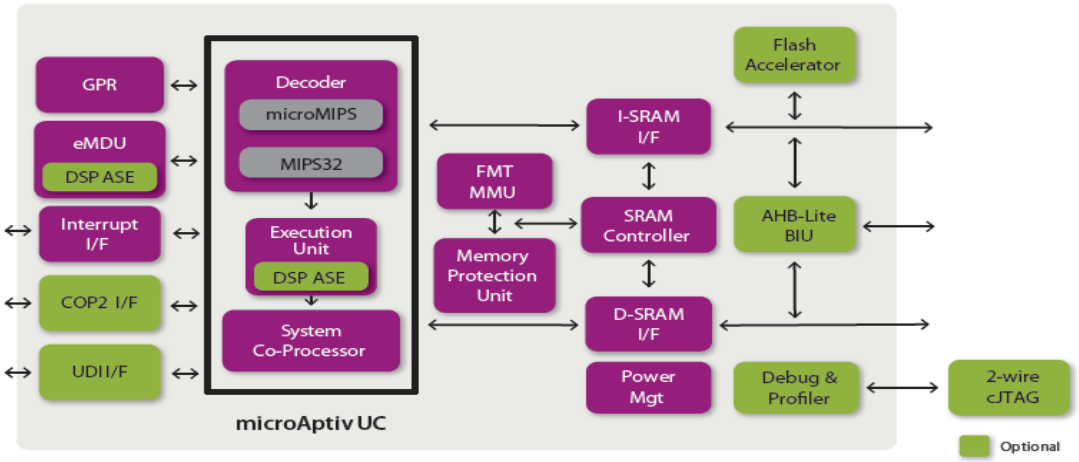
- **4KEc, most licensed MIPS Core**
  - >65 cumulative licensees
- **M4K, first MCU-specific MIPS Core**
  - >30 licensees, including Microchip, leading MCU developer
  - Shipping in MCU, Mobile Phone, Cable Modem, Consumer Electronics
- **M14K/c, fastest licensed MIPS Core**
  - Released Q1 2010 with >25 Licenses
  - Designed in MCU, Smart Grid, VoiP, Security, Wireless Network and Office Automation systems

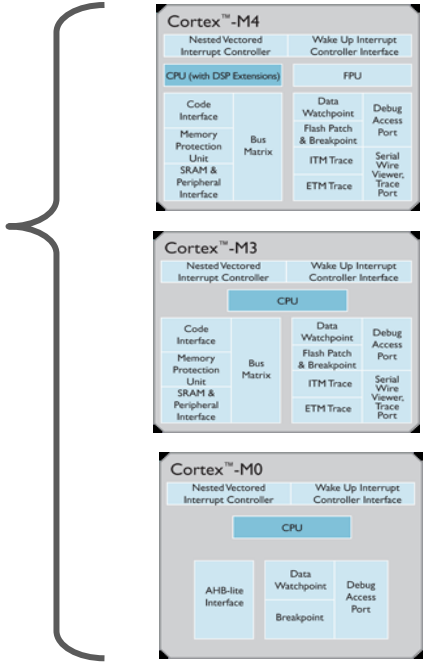
Winner: Best IP

Winner: Leading Product

- **microAptiv™ – Launched in Q2 2012**

Flexibility and Value – ‘Three Cores in One’





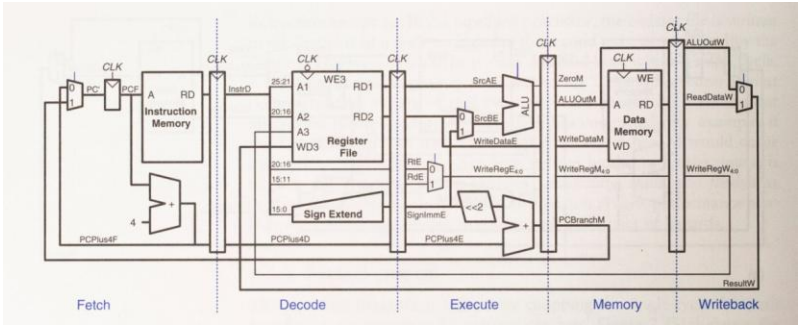
- microAptiv UC is highly configurable, equivalent to Cortex M0, M3 or M4
- One license, multiple cores, re-use capability
- Single design covers many applications
- Single development system
- Leverage design experience

Главные особенности

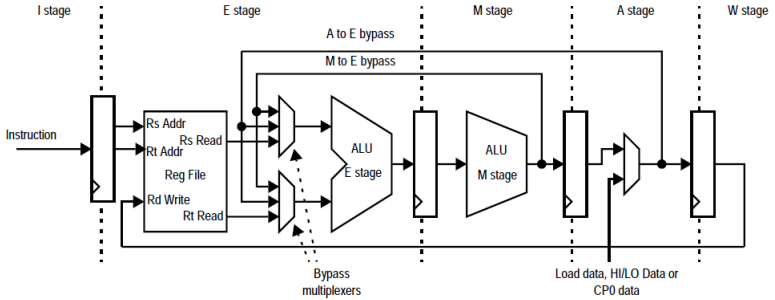
- **Пять стадий конвейера**
  - Использование форвардинга данных для минимизации остановок конвейера
- **32-битный набор инструкций и 16-битные инструкции для экономии памяти**
- **Два варианта трансляции виртуальных адресов для защиты памяти**
  - Фиксированная или с использованием буфера ассоциативной трансляции
- **Различные опции умножения и деления для разработчика SoC**
  - Быстрое и медленное, а также специальные команды для алгоритмов DSP
- **Векторные прерывания и поддержка внешнего контроллера прерываний**
- **Набор «теневых» регистров для ускоренной обработки прерываний**
  - Не требуется сохранение регистров в обработчике прерывания
- **Гибкий контроль энергопотребления**

Конвейер M4K напоминает конвейер из учебников

- **Сверху – конвейер процессора, реализующего подмножество архитектуры MIPS из учебника**
  - David Harris and Sarah Harris. Digital Design and Computer Architecture, 2-nd edition. 2012.

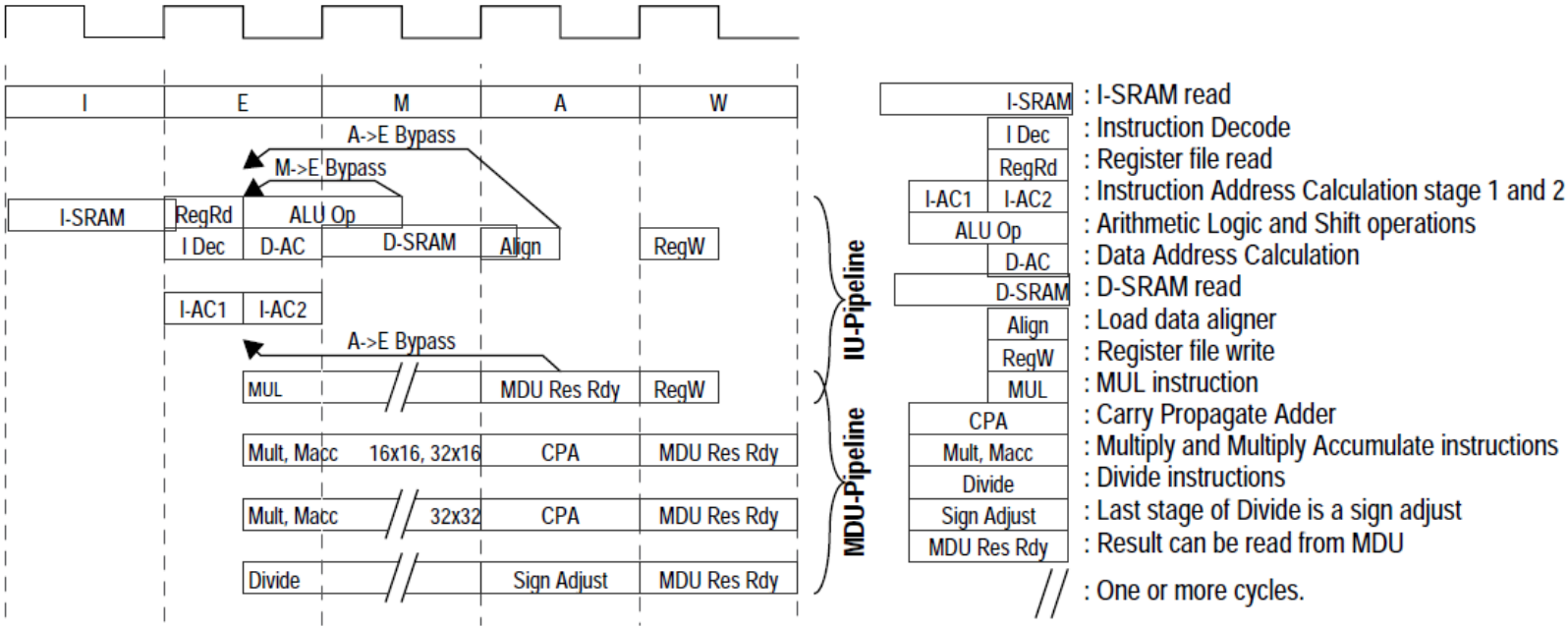


- **Снизу – конвейер индустриального процессора MIPS M4K**
  - MIPS32® M4K™ Processor Core Software User's Manual



Сохраняя преемственность от элегантного академического дизайна, индустриальный MIPS M4K оптимизирован по таймингу и содержит много опций

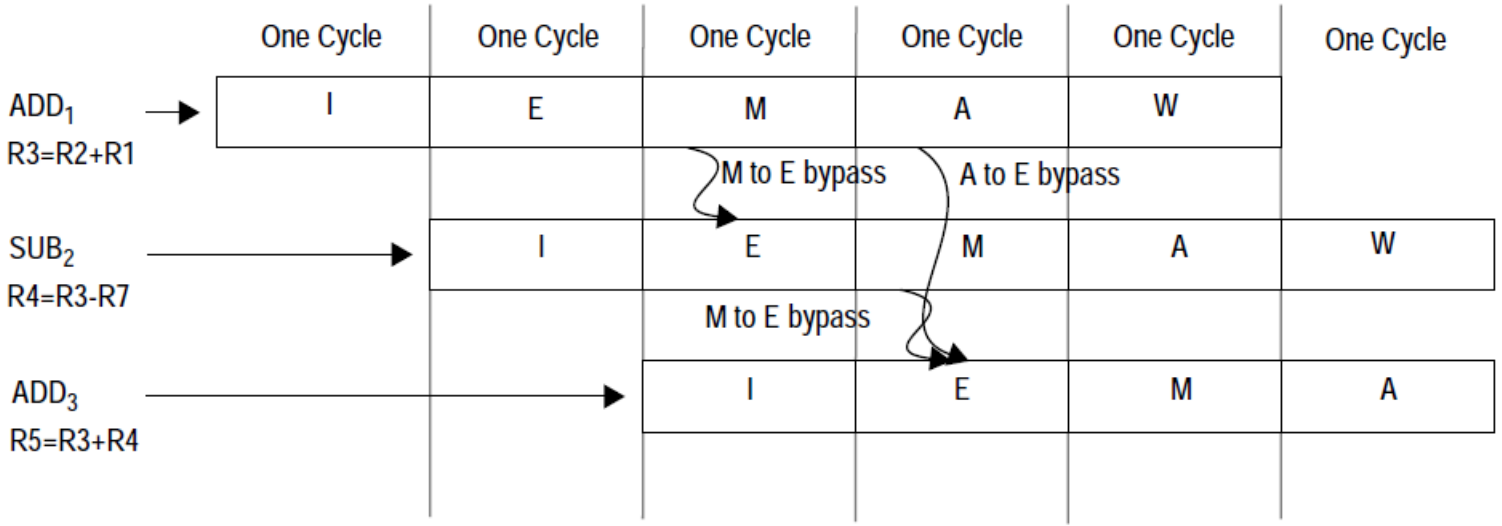
Более полная диаграмма конвейера MIPS M4K (вариант с быстрым умножением и делением)



Источник: MIPS32® M4K™ Processor Core Software User's Manual

Иллюстрация форвардинга в конвейере MIPS M4K

Форвардинг позволяет избежать остановок конвейера (stall и slip)



Источник: MIPS32® M4K™ Processor Core Software User's Manual

16-битные наборы инструкций – MIPS16e и microMIPS

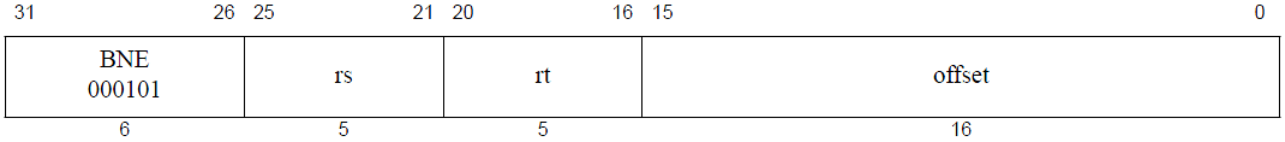
- **MIPS16e**
  - Используется в M4K и старших ядрах – 24K, 74K и других
  - Программы, скомпилированные с использованием MIPS16e – на 25-30% меньше, чем без него

- **microMIPS**
  - Реализован в M14K и microAptiv
  - Не просто расширение системы команд, а новая, альтернативная MIPS32 система команд, состоящая из смеси 16-ти и 32-битных команд
  - При «компрессии» 35% потеря быстродействия всего 2%

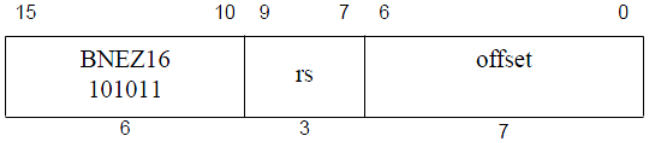
- **Переключения между режимами – на лету**
  - Главный способ переключения - переход или вызов функции по содержимому регистра, в младшем бите которого стоит 1

Пример 32-битной и 16-битной команд

MIPS32: Условный переход, когда содержимое двух регистров (*rs* и *rt)* не равно. Частный случай: условный переход, когда содержимое регистра не равно содержимому регистра 0, в котором всегда находится нуль.



microMIPS (M14K и interAptiv): Условный переход, когда содержимое регистра (*rs*) не равно нулю



Умножение и деление

- **Ядра M4K / M14K / interAptiv предоставляют разработчику системы на кристалле (System on Chip – SoC) несколько конфигураций ядра для умножения и деления**

  - Высокая производительность
    - Умножение за один цикл синхросигнала
    - Умножение со сложением (multiply-accumulate – MAC) за один или два цикла
      - 32-бита на 16-бит – за один цикл
      - 32-бита на 32 бита – за два цикла

  - Низкая производительность, зато и малая площадь на кристалле (и энергопотребление)
    - Итеративный алгоритм умножения

Зачем нужна специальная команда умножения со сложением - MADD?

- **Эта команда часто встречается в алгоритмах цифровой обработки сигналов – Digital Signal Processing (DSP)**
  - Например вот формула для простого частотного фильтра (Finite Impulse Response Filter – FIR filter), убирающего определенные частоты из оцифрованного звукового сигнала

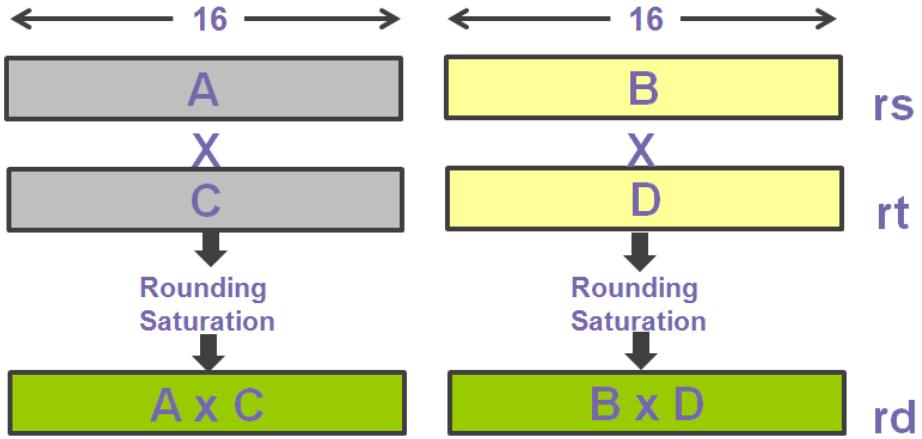
$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N]$$
$$= \sum_{i=0}^{N} b_i x[n-i]$$

microAptiv реализует набор инструкций для DSP

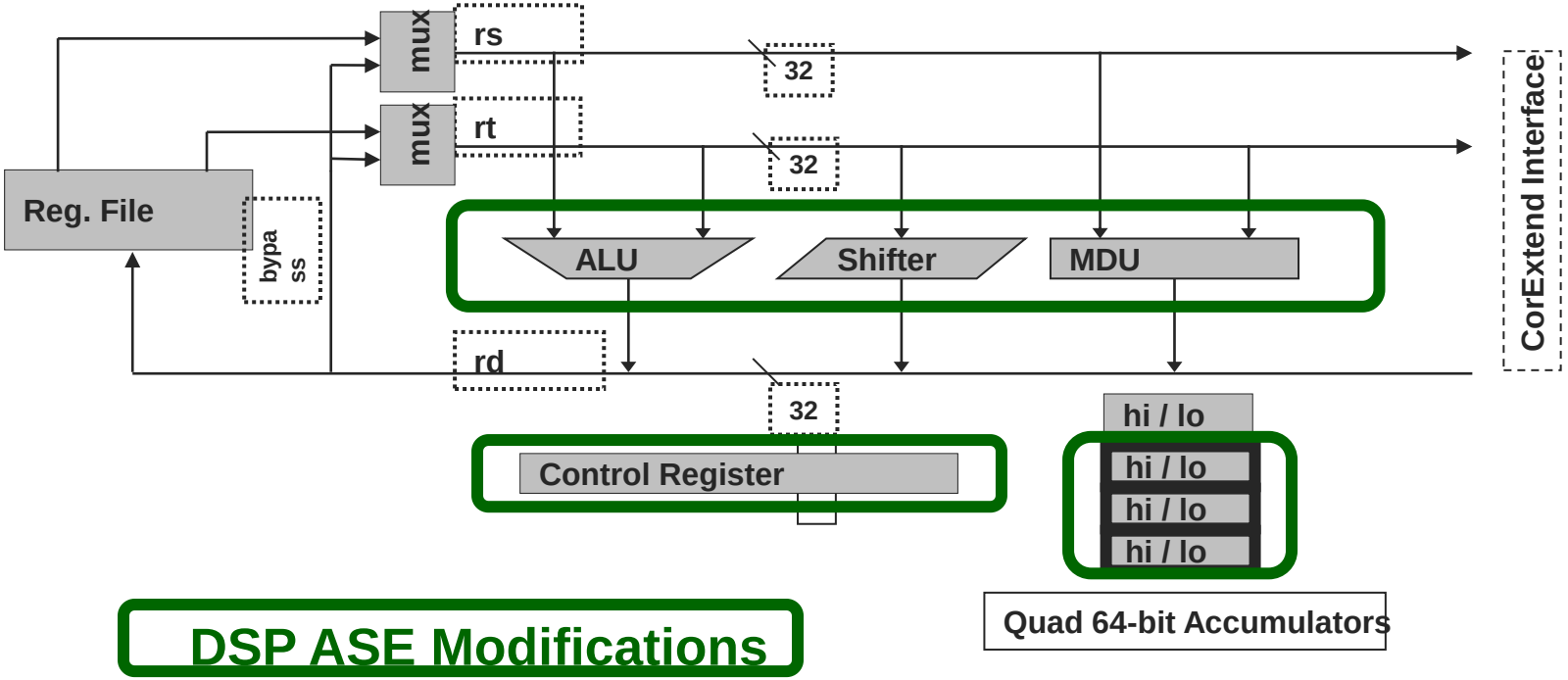
- **Инструкции для одновременных арифметических операций с четырьмя байтами 32-битного слова, которые рассматриваются как независимые числа**
  - То же – с двумя полусловами 32-битного слова
- **Арифметика с фиксированной точкой**
  - Для DSP алгоритмов наиболее полезными является 32-битные числа с точкой после старшего 31-го бита (Q31) и 16-битные числа с точкой после старшего 15-го бита (Q15). Старший бит и в одном, и в другом представлении содержит знак
- **Арифметика с насыщением – saturation arithmetic**
  - В этой арифметике есть понятие «много» и умножение или сложение любого числа с «много» дает «много»
- **Дополнительные операции умножения со сложением (multiply-accumulate – MAC), которые используют четыре независимых аккумулятора**
- **Операции округления, работы с битами и т.д. – все, что повышает бенчмарки у алгоритмов цифровой обработки сигналов**
- **Все эти инструкции могу использоваться с коде на C с помощью вызова специальных псевдо-функций**

Пример команды из DSP-расширения

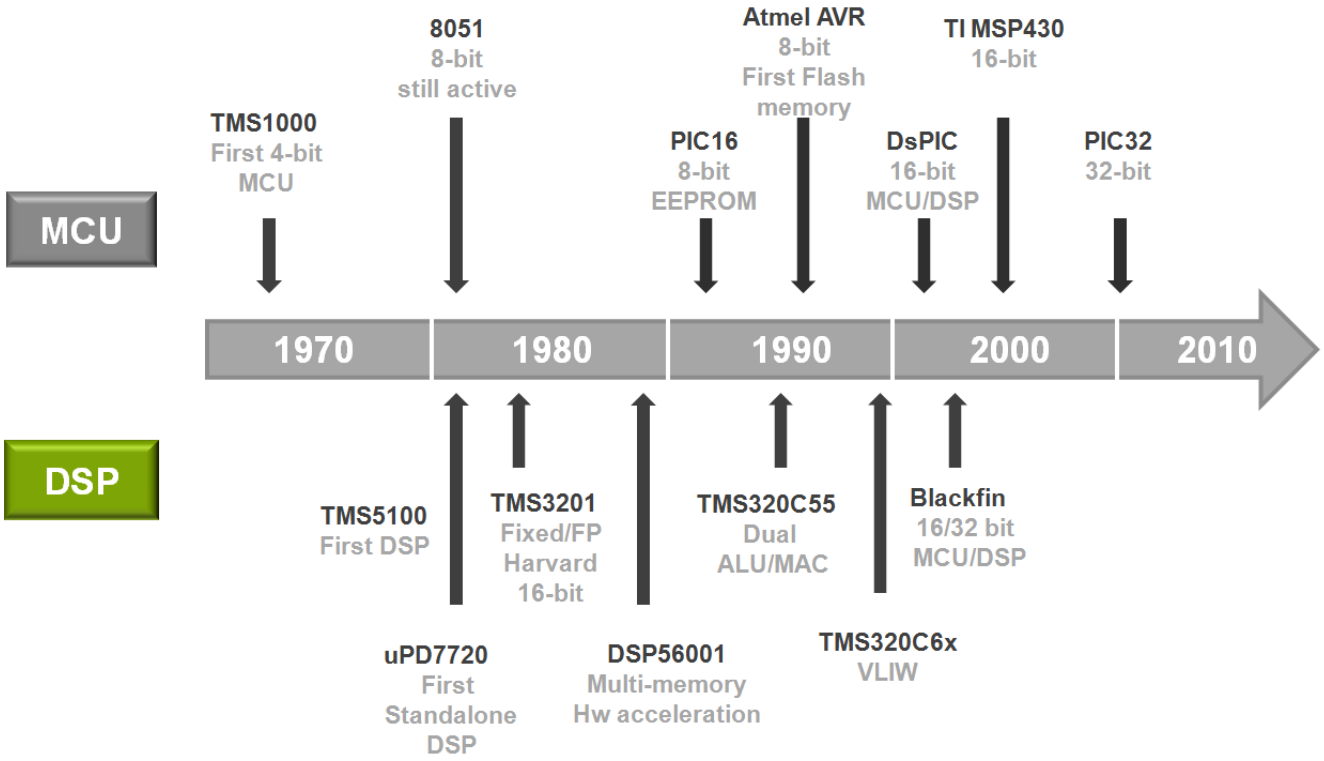
- **MULQ_RS.PH, rd, rs, rt**
  - Q означает «операция с фиксированной точкой» (fractional data type)
  - PH означает «независимо умножить 16-битные элементы двух 32-битных векторов»
  - RS означает «округление» (rounding) и «насыщение» (saturation)



DSP ASE Block Diagram



MIPS microAptiv - объединение двух трендов в эволюции микроконтроллеров и DSP



Новые инструкции для эксклюзивного доступа

- **В M14K и microAptiv появились новые инструкции для эксклюзивного доступа к памяти**
  - ASET – Atomic Bit Set
  - ACLR – Atomic Bit Clear
  - Инструкции работают только с некэшируемой (uncached) памятью

- **Что использовалось раньше в архитектуре MIPS для эксклюзивного доступа к памяти**
  - LL – Load Linked
  - SC – Store Conditional
  - Функциональность типа ASET требовала нетривиального программирования

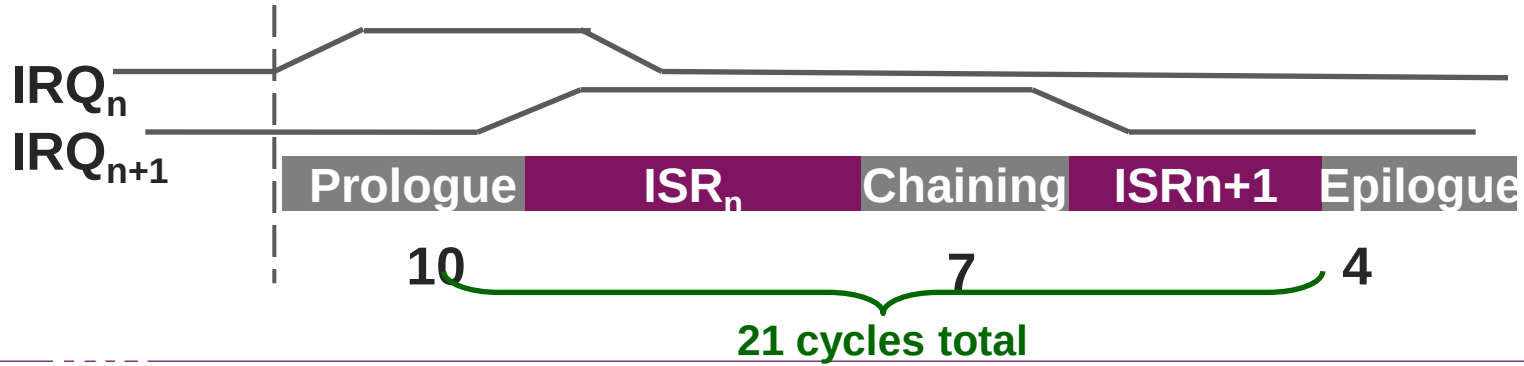
- **В чем преимущество новых инструкций?**
  - Гораздо проще писать код для частного случая эксклюзивного доступа к памяти
  - Работают с битами
  - Имеют предсказуемый тайминг для чтения, модификации и записи модифицированного значения

Оптимизация обработки прерываний в M14K и microAptiv

- Добавлено в M14K и соответственно в microAptiv
- Во время прерывания происходит спекулятивный prefetch для адреса обработчика прерывания
- Автоматическое сохранение в стеке и восстановление процессором регистра COP0 Status, EPS и подобной информации с Interrupt Automated Prologue (IAP) и Interrupt Automated Epilogue (IAE)
- «Цепные» (chained) прерывания – если одно прерывание случилось после другого, то первому не требуется возвращаться в код до прерывания – переход в обработчик второго случиться немедленно, даже минуя IAE и IAP
- Новая инструкция IRET в дополнение к старой ERET для использования с IAP/IAE и цепными прерываниями

MCU ASE - Reduced Interrupt Latency

- **Interrupt vector pre-fetching**
- **Automated interrupt prologue**
  - StackPtr adjustment – 3 to 32 words
- **Automated interrupt epilogue**
  - Provides options in dealing with nested exceptions
- **Interrupt chaining**



Экстра: Простор для инноваций в системах на кристалле

- **У внешнего интерфейса ядер M4K, M14K и microAptiv существует сигнал DS_Lock, который позволяет строить системы из очень большого количества малых ядер**
  - DS_Lock – индикатор доступа к памяти с помощью команд Load Linked (LL) и Store Conditional (SC)
  - LL и SC предназначены для программирования многоядерных систем
  - Теоретически разработчик системы на кристалле может посадить на одну микросхему сотни ядер M4K и сделать «суперкомпьютер на кристалле» для специализированных вычислительных задач

- **У всех этих ядер имеется интерфейс CorExtend для добавления блока «пользовательских» команд, а также интерфейс для «пользовательского» сопроцессора 2**
  - Под «пользователем» имеется в виду разработчик системы на кристалле
  - Интерфес для сопроцессора 2 использовался например для видеопроцессора Sony Playstation I и II
  - Об этих свойствах ядер более подробно рассказано в одной из следующих презентаций

For Distribution

Кэши в ядрах MIPS и микроконтроллерах PIC32

For Distribution

Кэш в MIPS microAptiv UP / Microchip PIC32MZ

Зачем нужны кэши?

- В 1960-е годы процессоры были медленнее, чем память
- С 1980-х годов скорость процессоров росла быстрее, чем скорость памяти
- За одно обращение к памяти современный процессор для десктопа может выполнить сотни арифметических инструкций; без кэша он будет простаивать
- Кэш полезен уже у PIC32MX (процессор - 80 MHz, флэш – 30 MHz)
- Для 200 MHz PIC32MZ кэш становится небходим

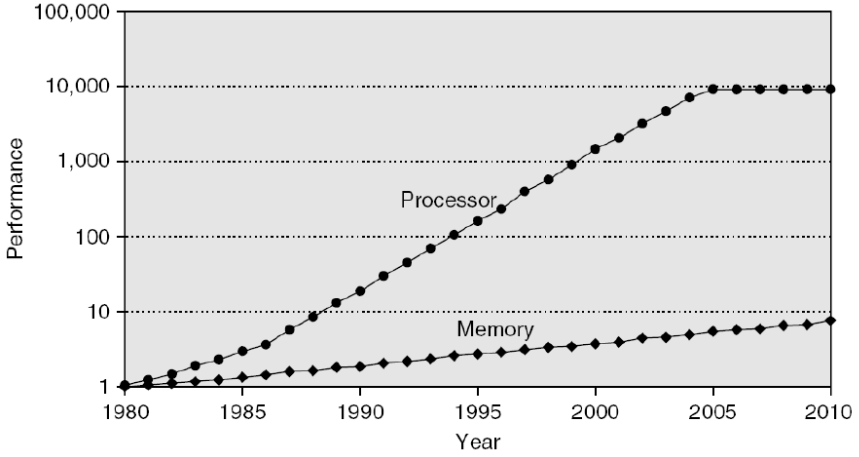


График из Hennessy & Patterson 2011,
через David Wentzlaff 2011 из Princeton University

Какие свойства программ использует кэш?

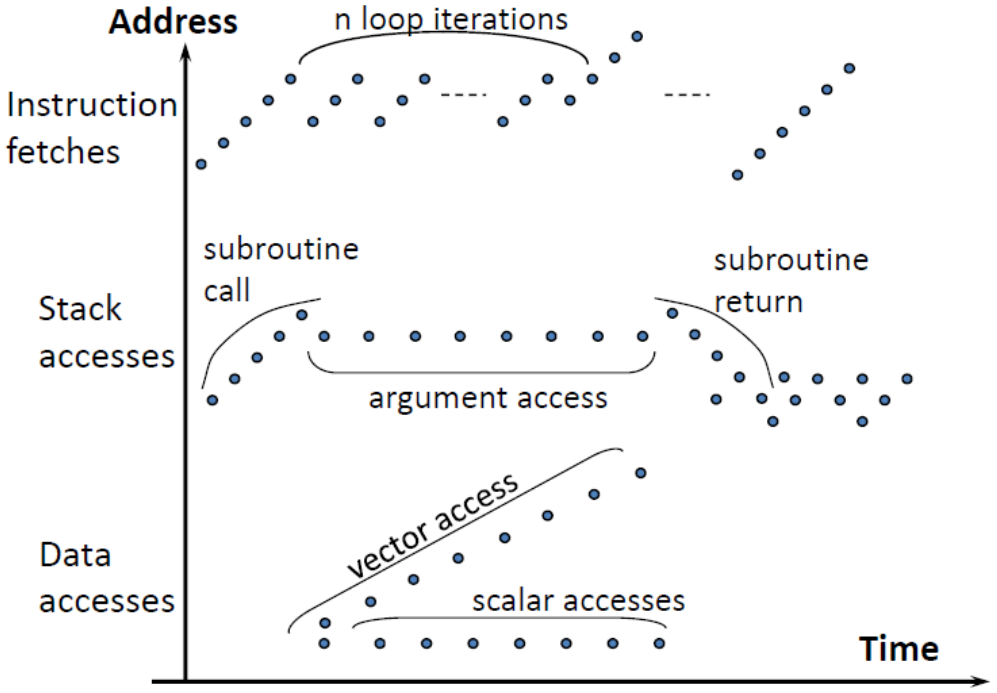


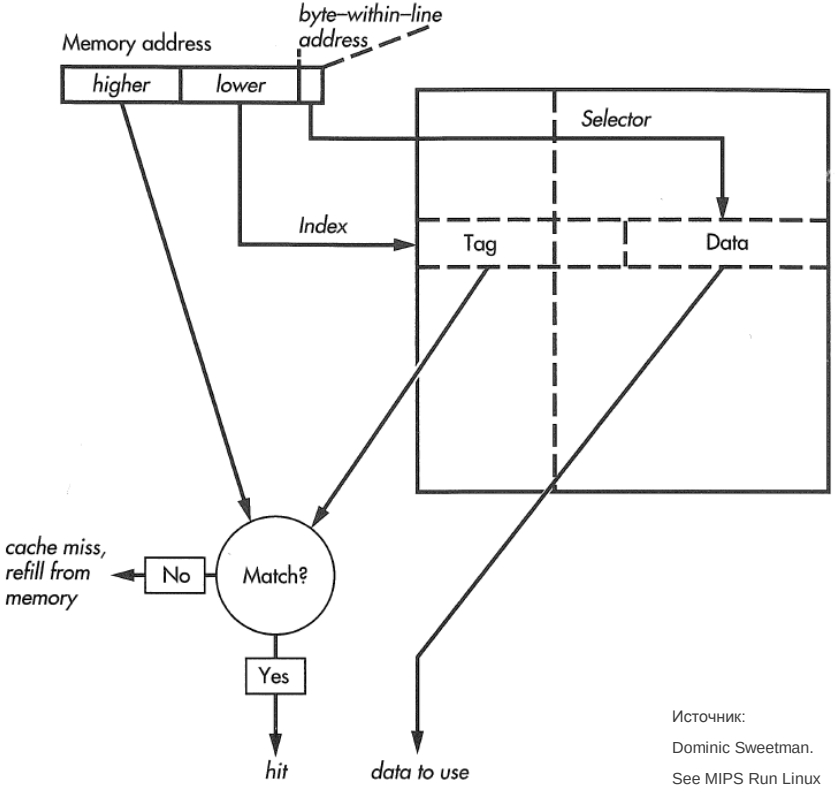
График из David Wentzlaff 2011, Princeton University

- **Временная локальность**
  - Если произошел доступ к адресу, он вероятно скоро повторится
- **Пространственная локальность**
  - Если произошел доступ к адресу, вероятно скоро будет доступ к соседнему адресу

Принцип работы кэша прямого отображения

Direct-mapped cache

- **Терминология**
  - Адрес — Address
  - Индекс — Index
  - Тэг — Tag
  - Строка — Line
  - Индекс байта — Byte index
  - Попадание — Hit
  - Промах — Miss
  - Вытеснение — Eviction

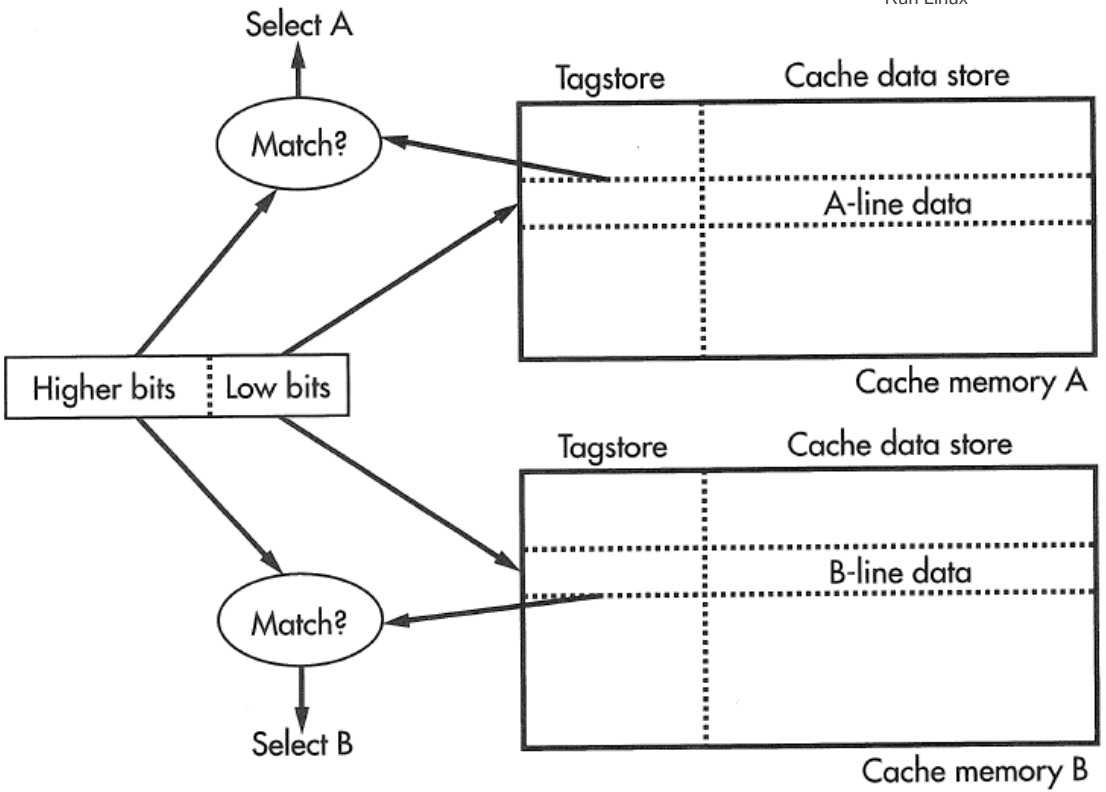


Источник:
Dominic Sweetman.
See MIPS Run Linux

Двухканальный множественно-ассоциативный кэш

Two-way set associative cache

- **Терминология (продолжение)**
  - Канал
    - Way
  - Политика замещения
    - Replacement policy
  - Наименее последний используемый
    - Least Recently Used (LRU)



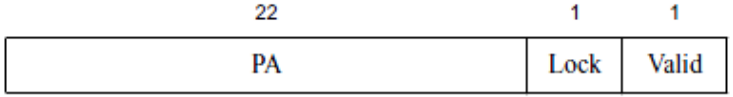
Источник: Dominic Sweetman. See MIPS Run Linux

Параметры кэшей в PIC32MZ

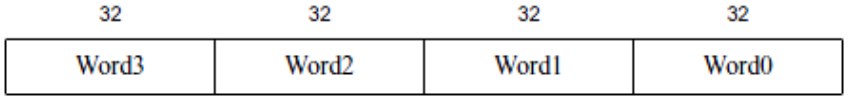
Частный случай конфигураций ядра MIPS microAptiv UP

- **L1 I-Cache – 16KB, 4-канальный множественно-ассоциативный кэш инструкций**
- **L1 D-Cache – 4KB , 4-канальный множественно-ассоциативный кэш данных**

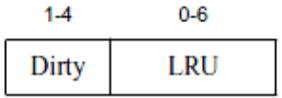
Tag (per way):



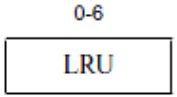
Data (per way):



D Way-Select:



I Way-Select:



Политика записи и выделения линии кэша

Write policy and write allocation

- **Политика сквозной записи**
  - Write-through
  - Любая запись в кэш (с попаданием или без) приводит к записи в память
- **Два варианта действия при промахе во время сквозной записи**
  - Выделять строку в кэше (write allocation) или нет
- **Политика отложенной записи**
  - Write-back
  - Иногда называют некорректно «обратная запись»
  - При попадании в кэш во время записи - запись в память не производится
  - Запись в память производится при вытеснении строки другой строкой (с другим тегом)
  - При политике отложенной записи выделение строки при промахе происходит всегда

Атрибуты кэшируемости и когерентности

Cache Coherency Attributes (CCA)

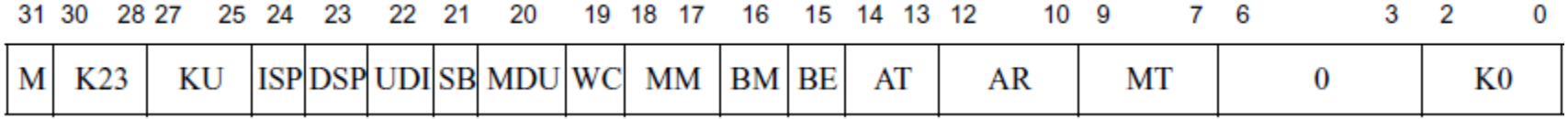
- **В PIC32MZ / MIPS microAptiv UP поддерживаются четыре аттрибута**
  - Uncached
  - Cacheable, noncoherent, write-through, no write-allocate
  - Cacheable, noncoherent, write-through, write-allocate
  - Cacheable, noncoherent, write-back, write-allocate

- **В PIC32MX / MIPS M4K поддерживаются только два аттрибута**
  - Uncached и cacheable
  - Используется для контроля внешнего (по отношению к M4K) кэша префетчера флэша

- **CCA может выставляться в двух местах ядер MIPS**
  - Если ядро сконфигурировано для MMU с TLB (случай PIC32MZ) – в атрибутах страницы
  - Если ядро сконфигурировано для MMU с FMT (случай PIC32MX) – в регистре Cop0 *Config*

Атрибуты кэшируемости и когерентности в Cop0 *Config*

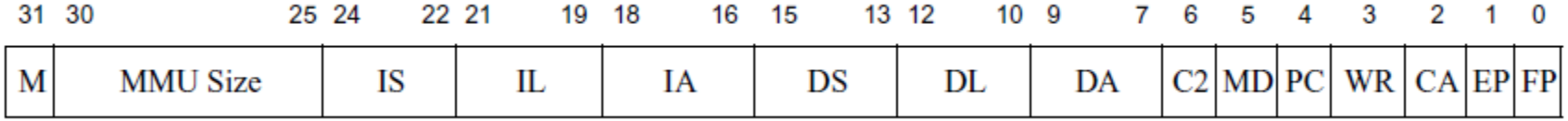
Частично относится к PIC32MX, не используется в PIC32MZ



- **Cop0 Config.K0 содержит атрибуты кэшируемости для сегмента kseg0**
  - 2 - uncached и 3 - cacheable
  - Включает внешний (по отношению к ядру MIPS M4K) кэша префетчера флэша
- **Другие подобные поля *Config***
  - KU для kuseg/useg
  - K23 для kseg2/kseg3 (не используется в PIC32)

Параметры кэшей в регистре Cop0 Config1

Полезно для написания кода, портабильного между несколькими ядрами MIPS



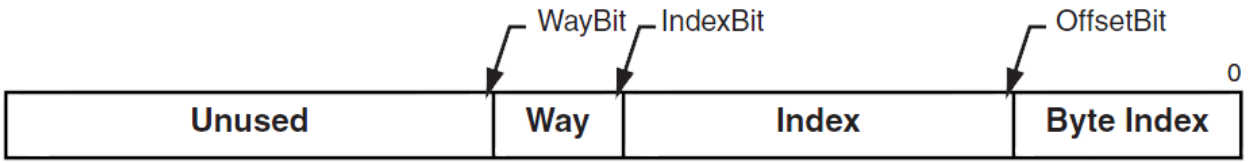
- **Cop0 Config1.IS – кодирует количество строк на канал в L1 кэше инструкций**
  - 0 – 64 строки, 1 – 128, 2 – 256, 3 – 512, 4 – 1024, кодировки 5-6 не используются
  - Кодировка 7 – 32 строки. Особый случай в других ядрах (не MIPS microAptiv UP)
- **IL – кодирует количество байтов в строке в L1 кэше инструкций**
  - 0 – кэш отсутствует, 3 – 16 байтов, 4 – 32 байта (в других ядрах), другие для L1 кэша инструкций не используются
- **IA - кодирует количество каналов в L1 кэше инструкций**
  - 0 – кэш прямого отображений, 1 – двух канальный, 2 – трехканальный, 3 – четырехканальный , другие кодировки для I-L1$ не используются
- **DS, DL, DA – аналогично для L1 кэша данных**
- **Для других ядер MIPS есть также параметры для L2 и L3 в регистре Config2**

Инструкция CACHE op, offset (base)

Для «ручного» управления кэшем

- ***op* состоит из кода операции с кэшем, а также кода кэша**
  - Код кэша – L1 для инструкций, L1 для данных, для других ядер также L2, L3
- **Коды операций с операндом-индексом**
  - Index Invalidate / Index Writeback Invalidate, Load Tag, Store Tag, Store Data
- **Коды операций с операндом-адресом**
  - Hit Invalidate, Fill / Hit Writeback Invalidate, Hit Writeback, Fetch and Lock

Формат операнда-индекса:



Инструкция CACHE – продолжение

Использование и регистры

- **Для инициализации кэшей нужно использовать CACHE StoreTag**
  - Поставить регистр Cop0 TagLo0.V (valid) = 0 и записать invalid tag во все строки
- **Для начального заполнения кэша инструкций можно делать CACHE Fill**
- **Для отладочной проверки состояния кэша CACHE LoadTag**
  - В зависимости от Cop0 ErrCtl.WST чтение будет происходить либо из Tag Array, либо из Way Select Array
  - Если Cop0 ErrCtl.WST= 0, тогда Cop0 TagLo0 будет содержать Tag, Valid, Dirty, Locked
  - Если Cop0 ErrCtl.WST= 1, тогда Cop0 TagLo0 будет содержать LRU
  - LRU = Least Recently Used. Отражает, какой канал использовался последним

Инструкция CACHE – продолжение

Другие случаи использования

- **До начала DMA из кэшируемой памяти – записать все из кэша**
  - CACHE HitWriteback или CACHE HitWritebackInvalidate
- **До начала DMA в кэшируемую память – убрать информацию из кэша**
  - CACHE HitInvalidate
- **Локировать линию – полезно для обработчиков прерываний**
  - CACHE Fetch and Lock

Инструкция CACHE – окончание

Использование

- **Очистить весь кэш – бывает нужно довольно редко**
  - CACHE IndexInvalidate
- **Для синхронизации кэша данных и кэша инструкций вместо последовательноси из CACHE можно использовать SYNCI offset(base)**
- **Для обычного чтения и записи кэш работает автоматически**
  - Инструкция CACHE – исключение, а не правило

Виртуально индексируемые и физически тэгируемые

Virtually Indexed and Physically Tagged (VIPT)

- **Свойство всех L1 кэшей ядер MIPS от MIPS Technologies**
- **Индекс берется от адреса до его трансляции MMU**
- **Тэг берется от адреса после его трансляции MMU**
- **Преимущество – скорость: кэш может работать параллельно с MMU**

- **Недостаток VIPT в некоторых ядрах - cache aliases**
  - Возникает, когда размер кэша для одного канала больше, чем размер страницы в MMU
  - Может привести к нескольким разным копиям данных для одного физического адреса
  - Требует специальных мер (программных и/или аппаратных) для корректной работы кэша

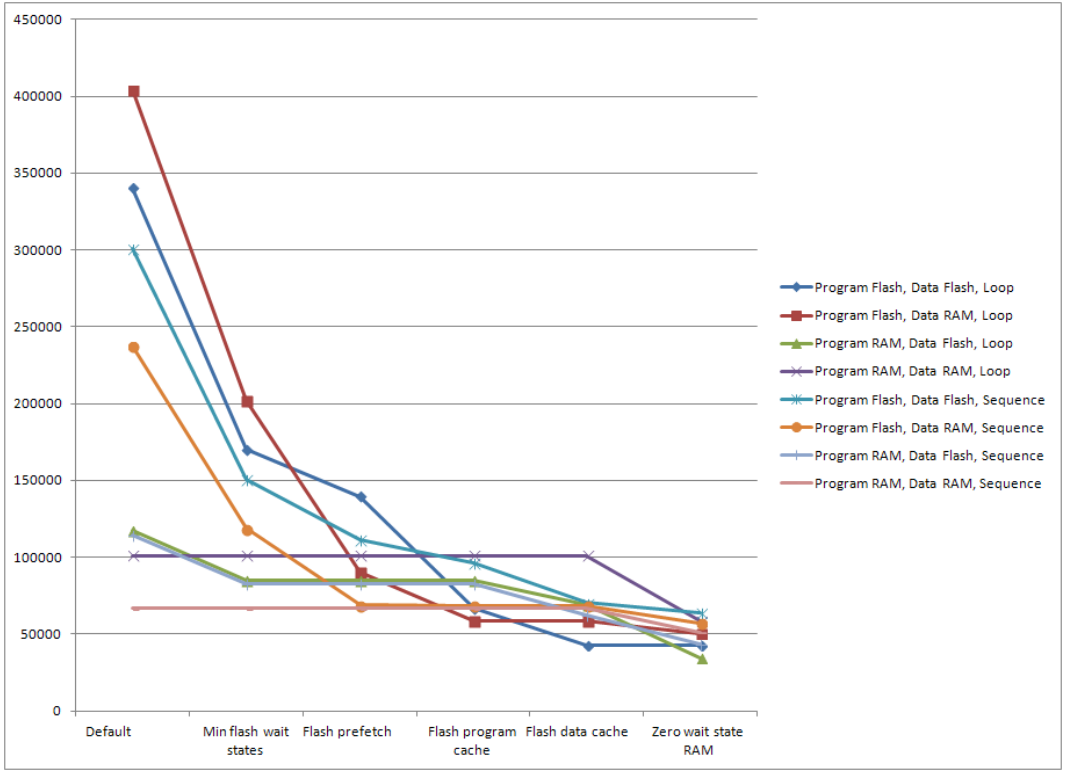
- **В PIC32MZ проблема cache aliases не возникает**

For Distribution

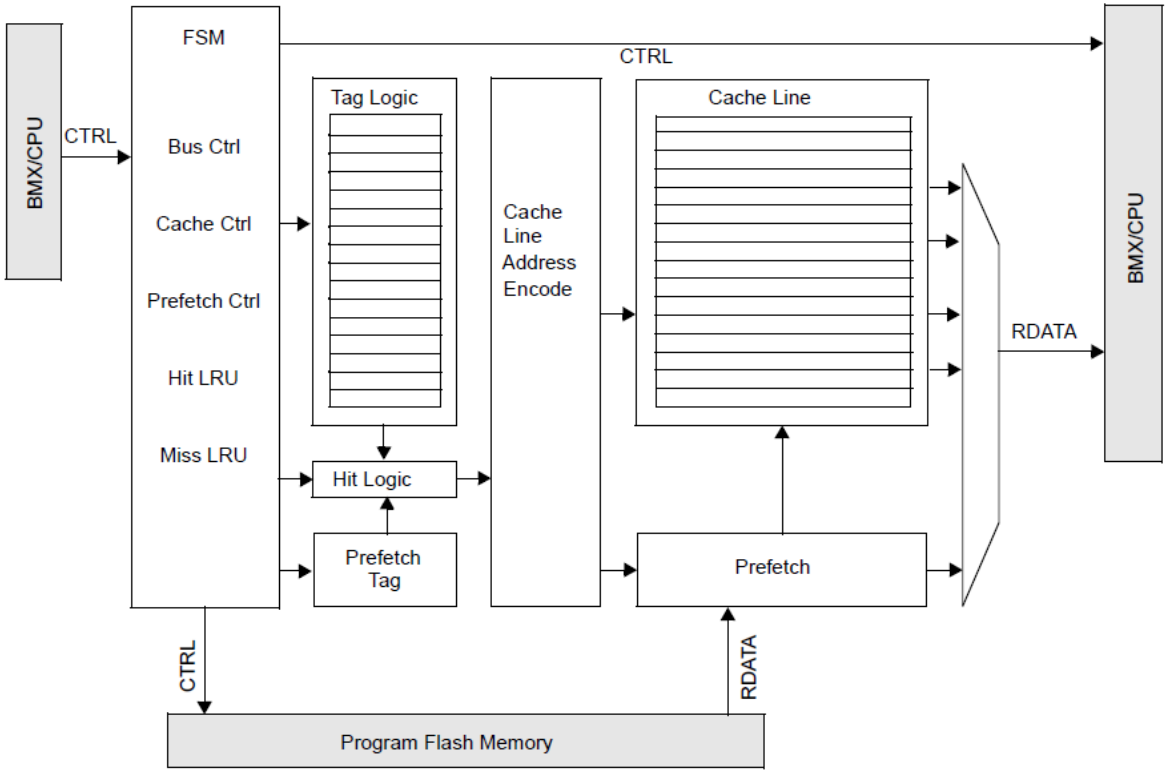
Кэш предварительной выборки в PIC32MX

Оптимизация доступа к флэшу и оперативной памяти, работа к кэшем и префетчером PIC32MX

- С помощью простой манипуляции контрольными регистрами PIC32MX и регистрами системного сопроцессора ядра MIPS M4K можно повысить скорость отдельных алгоритмов в 5-8 раз



Блок-схема кэша предварительной выборки PIC32MX



Пример программы, которую сильно ускоряет кэш

```
// По умолчанию функция находится во флэше
// (если нет ключевого слова __longramfunc__)

int calculate_sum (const int * a, int n, int m)
{
    int i, sum = 0;

    for (i = 0; i < n; i ++)  // Этот цикл
        sum += a [i];         // полностью помещается в кэш

    return sum;
}
```

Программа, которую ускоряет кэширование данных из флэша

```
const int array_in_flash [] // const указывает компилятору поместить массив во флэш
    = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };

// "volatile" стоит, чтобы компилятор не убирал «лишние» обращения к памяти от a []
int calculate_sum (const volatile int * a, int n, int m)
{
    int i, j, sum = 0;

    for (j = 0; j < m; j ++)
        for (i = 0; i < n; i ++)  // Массив используется много раз
            sum += a [i];         // и полностью помещается в 4 линии кэша
    return sum;
}
. . . . . .
    result = calculate_sum (array_in_flash, 16, 1000);
```

Кэш не ускоряет программу, которая расположена в RAM и оперирует данными из RAM

```
#include <sys/attribs.h>  // Этот header нужен для __longramfunc__

__longramfunc__ int calculate_sum (const int * a, int n, int m)
{
    int i, sum = 0;

    for (i = 0; i < n; i ++)
        sum += a [i];

    return sum;
}

// Также существует ключевое слово __ramfunc__
// для оптимального вызова из __longramfunc__ функций
```

Пример линейного кода без циклов, который сильно ускоряется префетчером

```
n +=   a [ 0] + a [ 1] + a [ 2] + a [ 3]
     + a [ 4] + a [ 5] + a [ 6] + a [ 7]
     + a [ 8] + a [ 9] + a [10] + a [11]
     + a [12] + a [13] + a [14] + a [15];
```

Контрольные регистры кэша PIC32MX

- **CHEACC**
  - Номер линии
- **CHETAG**
  - Состояние линии – адрес, LVALID, LLOCK, LTYPE
- **CHEMSK**
  - Для кэширования начала нескольких обработчиков прерываний
- **CHEW0-CHEW3**
  - Линия как таковая
- **Статистика**
  - CHEHIT, CHEMIS и CHEPFABT

Figure 4-3: Mask Line

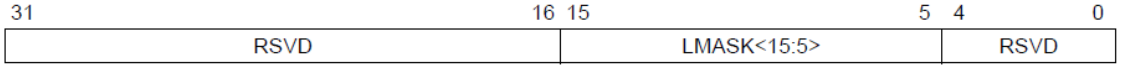


Figure 4-4: Tag Line

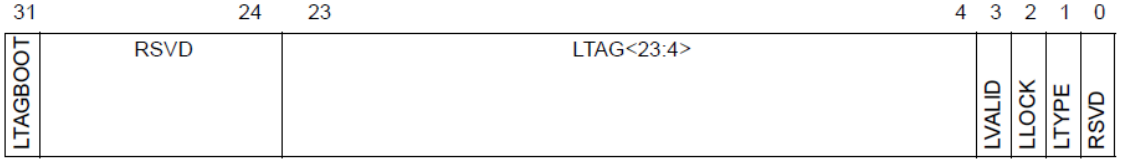
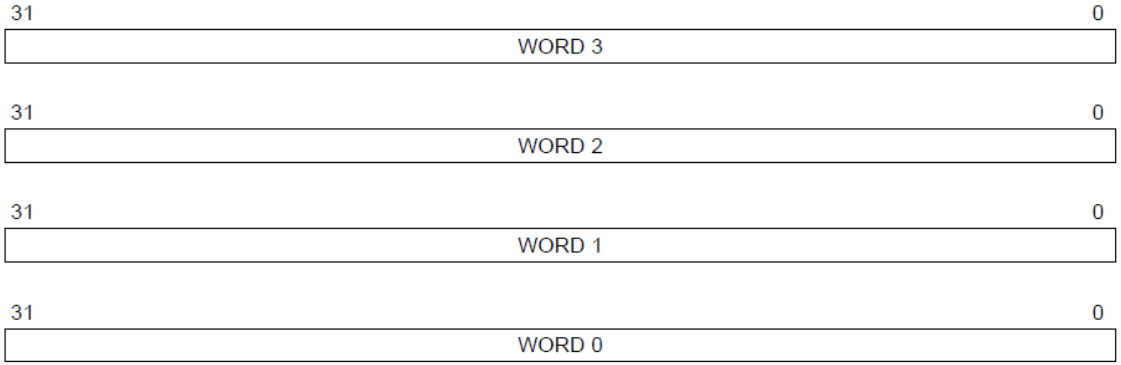


Figure 4-5: Data Line



Как включить кэш PIC32MX – два способа

- С помощью библиотечной функции **CheKseg0CacheOn ()** из **plib.h**
  - Подробно про «сегмент Kseg0» см. документацию по архитектуре MIPS

- Альтернативно - с помощью манипуляции регистрами системного сопроцессора (Coprocessor 0) ядра MIPS M4K

```
#include <xc.h>
. . . . .
unsigned config = _mfc0 (_CP0_CONFIG, _CP0_CONFIG_SELECT);
config &= ~ _CP0_CONFIG_K0_MASK;
config |= 3 << _CP0_CONFIG_K0_POSITION;
_mtc0 (_CP0_CONFIG, _CP0_CONFIG_SELECT, config);
```

Как включить кэширование данных флэша PIC32MX

- **CHECONbits. DCSZ = 3;**
  - Выделить 4 линии кэша для данных
- **CHECONbits. DCSZ = 2**
  - Выделить 2 линии кэша для данных
- **CHECONbits. DCSZ = 1**
  - Выделить 1 линию кэша для данных
- **CHECONbits. DCSZ = 0**
  - Запретить кэширование данных флэша

Как включить префетчер PIC32MX

- **CHECONbits.PREFEN = 3**
  - включить для кэшируемой и некешируемой области флэша
- **CHECONbits.PREFEN = 2**
  - включить только для некешируемой области флэша
- **CHECONbits.PREFEN = 1**
  - включить только для кешируемой области флэша
- **CHECONbits.PREFEN = 0**
  - выключить префетчер

Ошибки при работе с кэшем PIC32MX

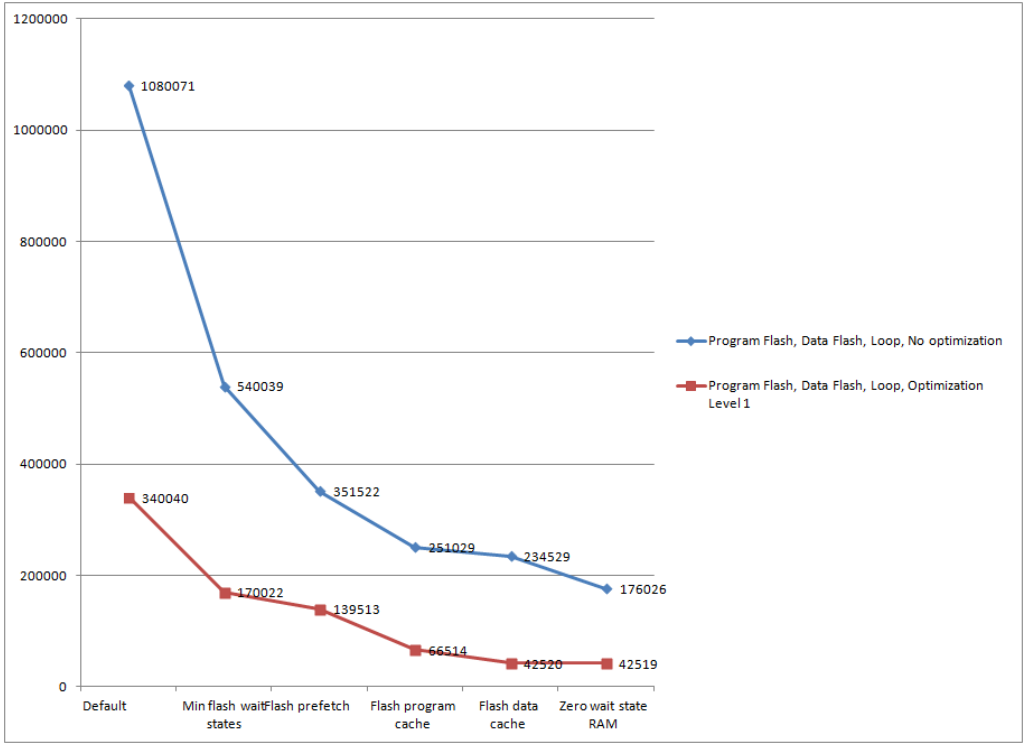
- Ошибка: локировать несколько линий с одним адресом
- Ошибка: читать линии кэша кодом, который сам кэшируется

Другие полезные оптимизации доступа к флэшу и RAM в PIC32MX

- **Снижение циклов ожидания доступа к флэшу**
  - По умолчанию CHECONbits.PFMWS = 7 циклов
  - Частота флэша 30 MHz. Когда частота процессора 80 MHz, 3 циклов ожидания достаточно
  - CHECONbits.PFMWS = (SYSCLK_FREQUENCY + FLASH_FREQUENCY - 1) / FLASH_FREQUENCY;
- **Удаление цикла ожидания доступа к RAM**
  - Этот цикл нужен для отладки,но если останов по доступу не нужен, его можно убрать
  - Первый способ - mBMXDisableDRMWaitState ()
  - Второй способ - BMXCONbits.BMXWSDRM = 0;

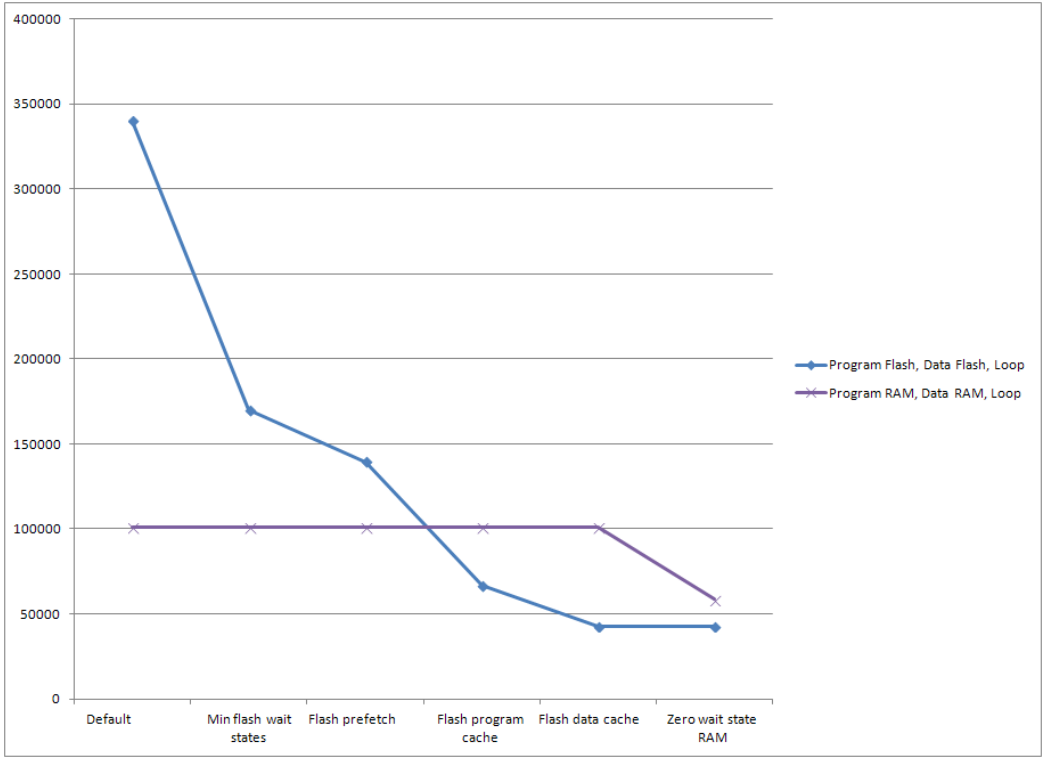
Перед оптимизацией доступа к памяти стоит включить оптимизации компилятора

- Неоптимизированный компилятором код делает слишком много обращений к памяти стека, на котором хранятся временные переменные



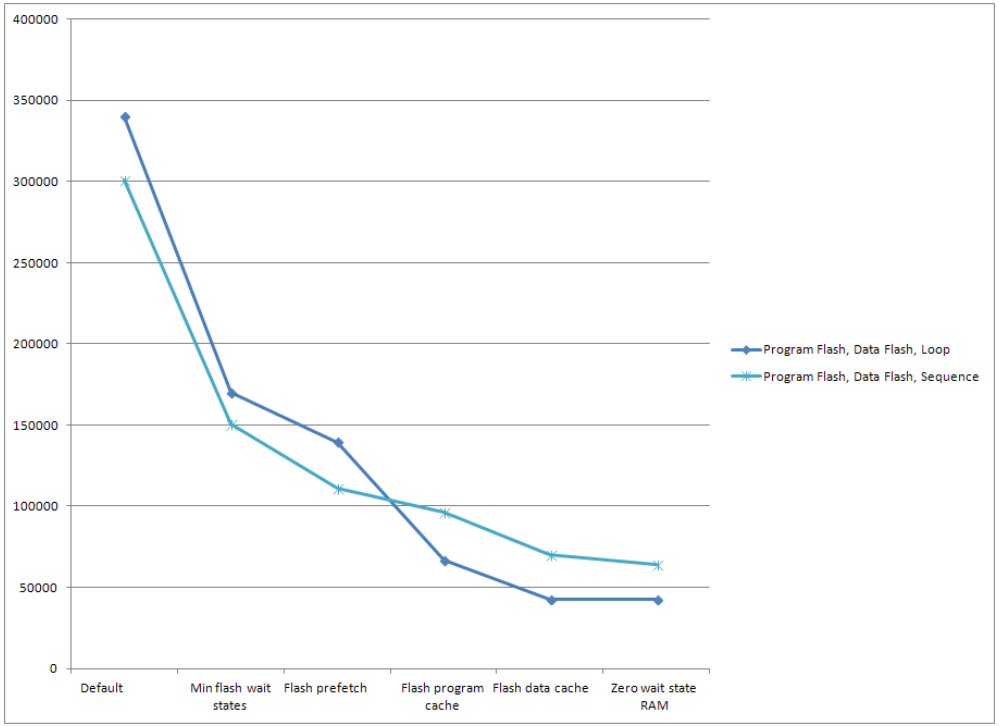
Флэш с включенными оптимизациями может работать быстрее RAM

- Снижение циклов ожидания флэша ускорило тест вдвое
- Префетч ускорил тест еще на 18%
- Кэш ускорил тест еще вдвое
- Кэширование данных ускорило еще на треть
- Следует помнить, что тест - искусственный



Влияние оптимизаций зависит от типа кода

- Кэш хорошо ускоряет компактные циклы
- Префетчер хорошо ускоряет линейные последовательности инструкций
- Сочетание масок и локированных линий можно использовать для ускорения нескольких обработчиков прерываний
- Кэширование данных может работать и в плюс, и в минус



Чтобы поставить все эти оптимизации

- SYSTEMConfigPerformance – см документацию

Пример оптимизации доступа к памяти и флэшу, работы к кэшем и префетчером

- **Пример находится на Google Code, можно копировать**
  - http://code.google.com/p/pic32-examples/source/browse/trunk/showroom/pic32mx_prefetch_cache
- **Тест для анализа полезности префетчера и префетч-кэша PIC32MX**
  - http://code.google.com/p/pic32-examples/source/browse/trunk/showroom/pic32mx_prefetch_cache/main.c
- **Утилита для распечатки состояния префетч-кэша PIC32MX**
  - http://code.google.com/p/pic32-examples/source/browse/trunk/showroom/pic32mx_prefetch_cache/prefetch_cache.c

For Distribution

Устройство управления памятью в процессорах MIPS

www.imgtec.com

Общие задачи устройства управления памятью

Memory Management Unit (MMU)

- **Предоставить доступ программы к объему виртуальной памяти, который превосходит объем физической оперативной памяти**
  - Рассматривалась как неактуальная задача для микроконтроллеров
  - Становится теоретически реализуемо с PIC32MZ
- **Ограничить доступ пользовательских программ к памяти операционной системы**
  - Полезно для некоторых систем со встроенными процессорами
- **Защитить пользовательские программы друг от друга**
  - Полезно для некоторых систем со встроенными процессорами

Два способа реализации MMU в архитектуре MIPS

- **Гибкий, используя так называемый буффер ассоциативной трансляции**
  - Translation Lookaside Buffer (TLB)
  - Позволяет защитить не только операционную систему от пользовательских программ, но и пользовательские программы друг от друга
- **Фиксированный**
  - Fixed Mapping Translation (FMT)
  - Требует минимум аппаратных ресурсов
- **И с FMT, и с TLB защита достигается трансляцией виртуальных адресов в физические с исключением в случае доступа пользовательской программы к запрещенным для нее адресам**

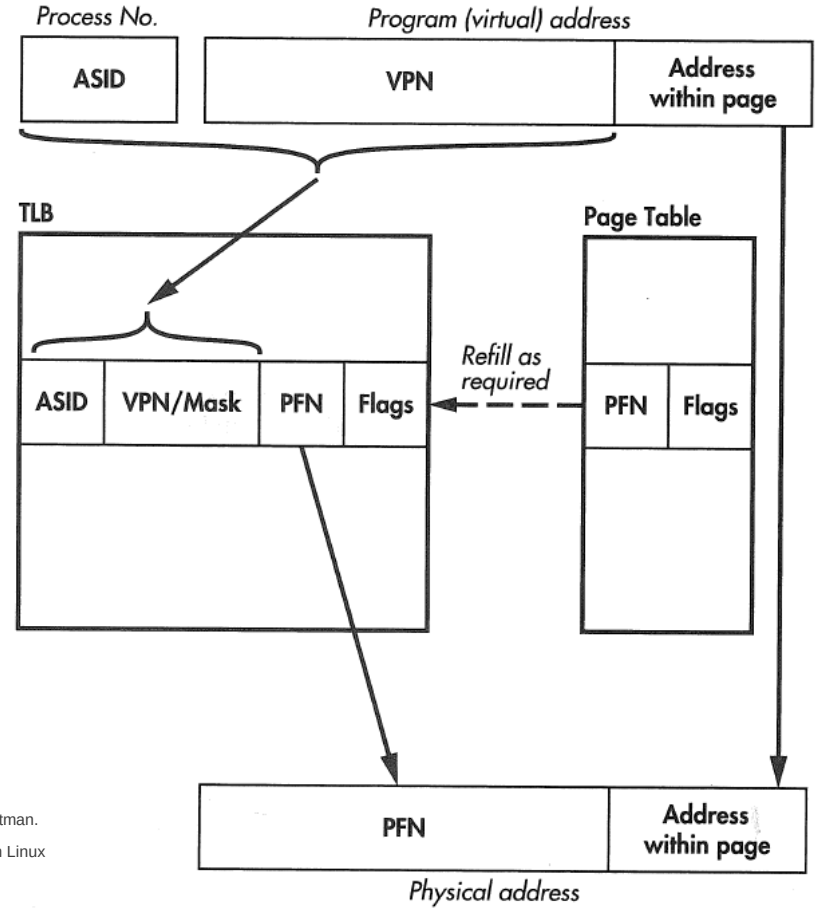
For Distribution

Буффер ассоциативной трансляции
Translation Lookaside Buffer (TLB)

Что такое TLB?

Translation Lookaside Buffer

- **Буффер ассоциативной трансляции**
- **На входе**
  - Виртуальный адрес, Virtual Address
    - Используется программой
  - Идентификатор адресного пространства
    - Address Space Identifier, ASID
- **На выходе**
  - Физический адрес, Physical Address
    - Используется контроллером памяти



Источник:

Dominic Sweetman.

See MIPS Run Linux

Что позволяет TLB?

Универсальное средство быстрого преобразования адресов

- **Позволяет скрывать память операционной системы от непривилегированного кода**
- **Позволяет нескольким регионам памяти выглядеть как последовательный кусок**
- **Позволяет разместить программу в любой части физической памяти**
- **Позволяет адресовать бОльшую по размеру физическую память, чем доступно виртуальных адресов**
- **Позволяет подгружать куски программы с внешнего устройства по надобности**

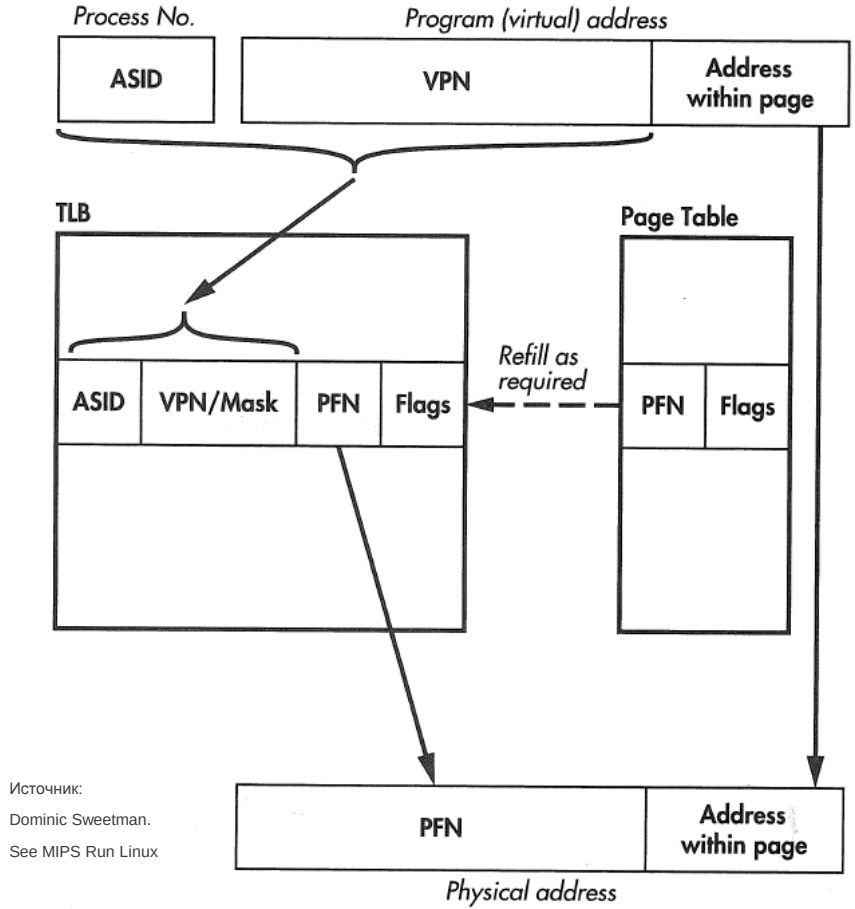
Что еще позволяет TLB?

TLB также ассоциирует с адресом различные атрибуты

- **Атрибут запрета чтения**
- **Атрибут запрета записи**
  - Альтернативно - индикатор, что на страницу с данным адресом происходила запись
- **Атрибут запрета исполнения**
- **Атрибуты кэшируемости и когерентности**

Страницы TLB

- **Память разбивается на страницы**
- **Страница может иметь размеры 4KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB**
  - Особый случай – 1KB (не поддерживается в PIC32MZ)
  - Разные страницы могут иметь разный размер
- **Номер виртуальной страницы**
  - Virtual Page Number, VPN
  - Виртуальный адрес = VPN + адрес внутри страницы
- **Номер физического фрейма**
  - Physical Frame Number, PFN
  - Физический адрес = PFN + адрес внутри страницы

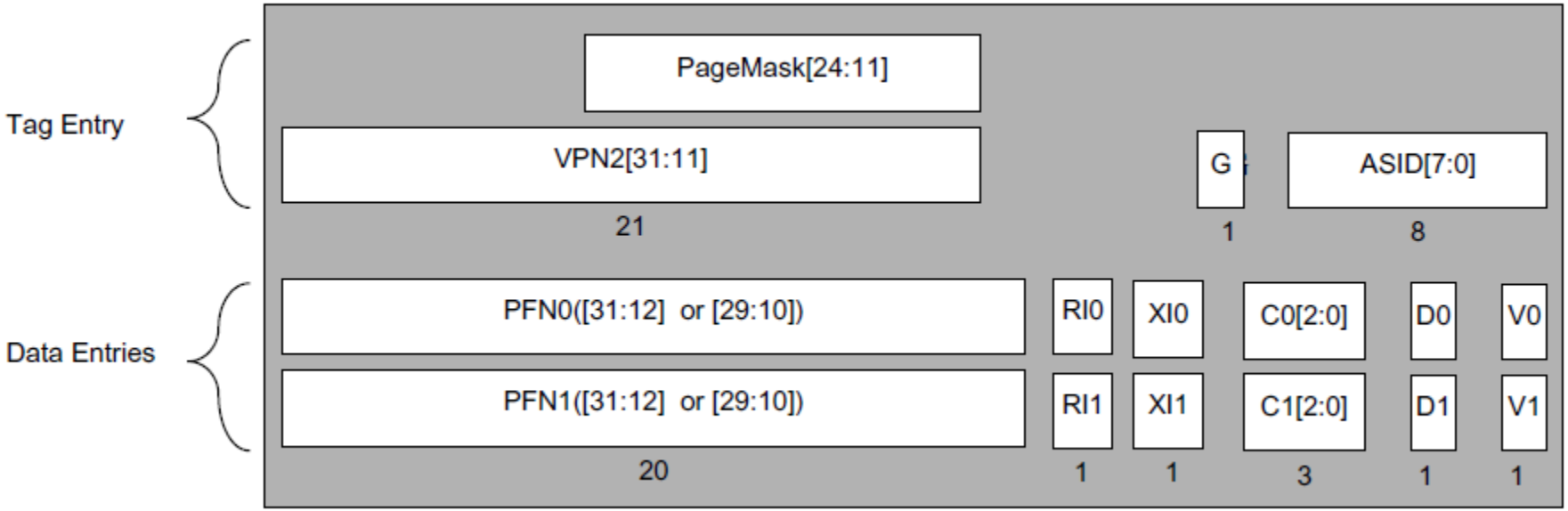


Источник:
Dominic Sweetman.
See MIPS Run Linux

Запись трансляции в TLB MIPS microAptiv UP

TLB Entry

- Делится на запись тега и две записи данных
- Описывает две трансляции – четной и нечетной страницы



Размер TLB

- **Ассоциативная память очень дорогая**
  - Content-addressable memory (CAM)
  - К каждой записи требуется компаратор
- **Типичный TLB в MIPS – от 16 до 64 трансляций**
- **Трюк с двойными записями родился для экономии на компараторах**
  - Согласно Майку Гупте, одного из дизайнеров MIPS R4000 (1991) :
  - Процессор с 64 одинарными записями TLB не влазил в floorplan
  - Влез дизайн с 24 двойными записями (48 трансляций)
  - Решение стало стандартным в начале 1990-х по требованию Микрософта
    - Микрософт и MIPS были членами комитета Advanced Computing Environment (ACE)
    - Микрософт портировал на MIPS Windows NT
  - В 1999 году трюк с двойными записями просочился в стандарт MIPS32

Значение полей тега в записи TLB

- **VPN2 [31:13] – Virtual Page Number**
  - Номер виртуальной страницы, деленный на 2
- **PageMask [28:11]**
  - Фактически определяет размер страницы, маскируя определенные биты виртуального адреса, чтобы они не участвовали в сравнении
- **ASID [7:0] – Address Space Identifier**
  - Идентифицирует процесс, тред или другой примитив операционной системы, для которого делается эта трансляция
- **G – Global Bit**
  - Трансляция является «глобальной», поле ACID игнорируется

Поле PageMask - подробнее

- **PageMask – фактически определяет размер страницы, маскируя определенные биты виртуального адреса, чтобы они не участвовали в сравнении:**

PageMask	Page Size	Even/Odd Bank Select Bit
00_0000_0000_0000_0000	1KB	VAddr[10]
00_0000_0000_0000_0011	4KB	VAddr[12]
00_0000_0000_0000_1111	16KB	VAddr[14]
00_0000_0000_0011_1111	64KB	VAddr[16]
00_0000_0000_1111_1111	256KB	VAddr[18]
00_0000_0011_1111_1111	1MB	VAddr[20]
00_0000_1111_1111_1111	4MB	VAddr[22]
00_0011_1111_1111_1111	16MB	VAddr[24]

Значение полей данных в записи TLB - 1

- **PFN0, PFN1 ([31:12] или [29:10]) – Physical Frame Number**
  - Верхние биты физического адреса для четной и нечетной страницы
  - [29:10] – для конфигурации ядра MIPS, которая позволяет страницы в 1KB
    - Конфигурация MIPS microAptiv UP для PIC32MZ не позволяет страницы в 1KB; 4KB минимум
  - Для страниц больше 4KB используется только часть битов
- **V0, V1 - Valid**
  - Если не установлен, попытка доступа к странице вызывает исключение TLB Invalid

Значение полей данных в записи TLB - 2

- **RI0, RI1 – Read Inhibit**
  - Вызывает прерывание при попытке чтения страницы
  - Для работы требует, чтобы Cop0 PageGrain.RIE = 1 (Read Inhibit Enable)
- **XI0, XI1 – Execute Inhibit**
  - Вызывает прерывание при попытке чтения страницы для исполнения (fetch)
  - Для работы требует, чтобы Cop0 PageGrain.XIE = 1 (Execute Inhibit Enable)
- **D0, D1 – “Dirty” («Грязный») или Write Enable**
  - Если бит установлен, то на страницу можно писать
  - Если бит не установлен, то происходит исключение TLB Modified
  - Обработчик исключения может установить бит и при вытеснении страницы из TLB записать ее содержимое куда-нибудь (применяется для «больших» операционных систем)

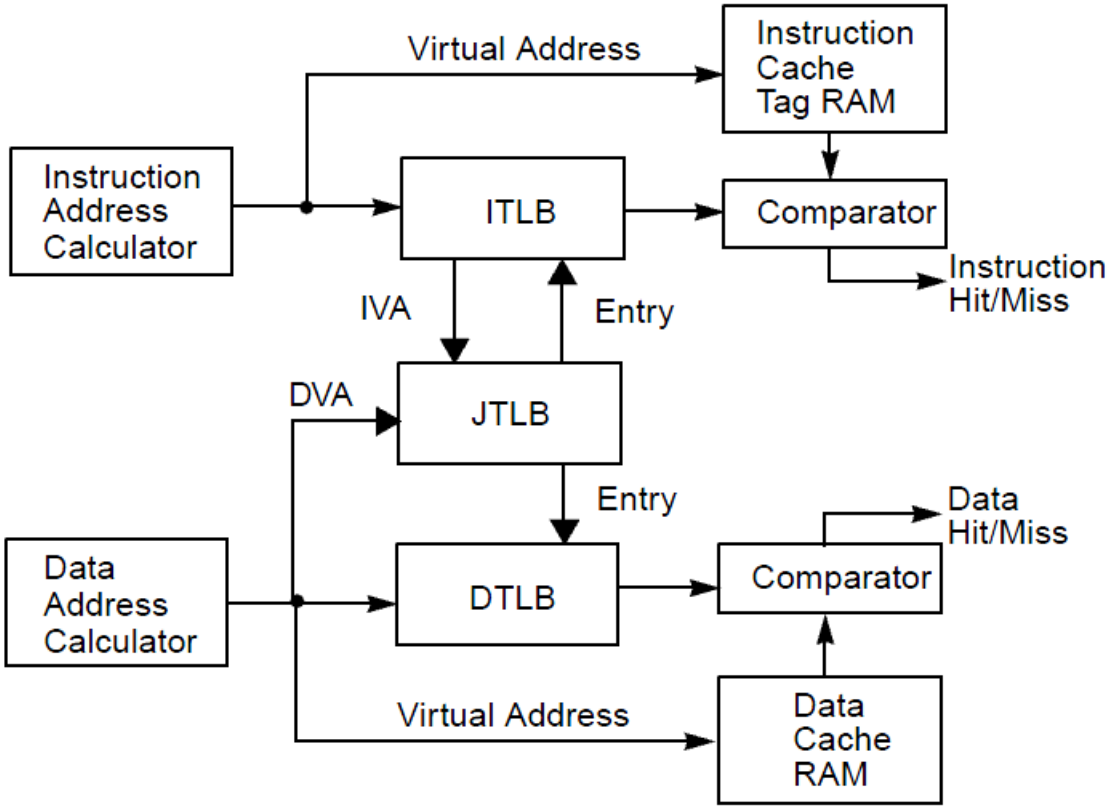
Значение полей данных в записи TLB - 3

- **C0 [2:0], C1 [2:0]**
  - Cacheability and Coherence Attributes (CCA)
  - Атрибуты кэшируемости и когерентности

C[2:0]	Coherency Attribute
000	Cacheable, noncoherent, write-through, no write-allocate
001	Cacheable, noncoherent, write-through, write-allocate
010	Uncached
011	Cacheable, noncoherent, write-back, write-allocate
100	Maps to entry 011b*
101	Maps to entry 011b*
110	Maps to entry 011b*
111	Maps to entry 010b*
* These mappings are not used on the microAptiv UP processor cores but do have meaning in other MIPS Technologies implementations. Refer to the MIPS32 specification for more information.	

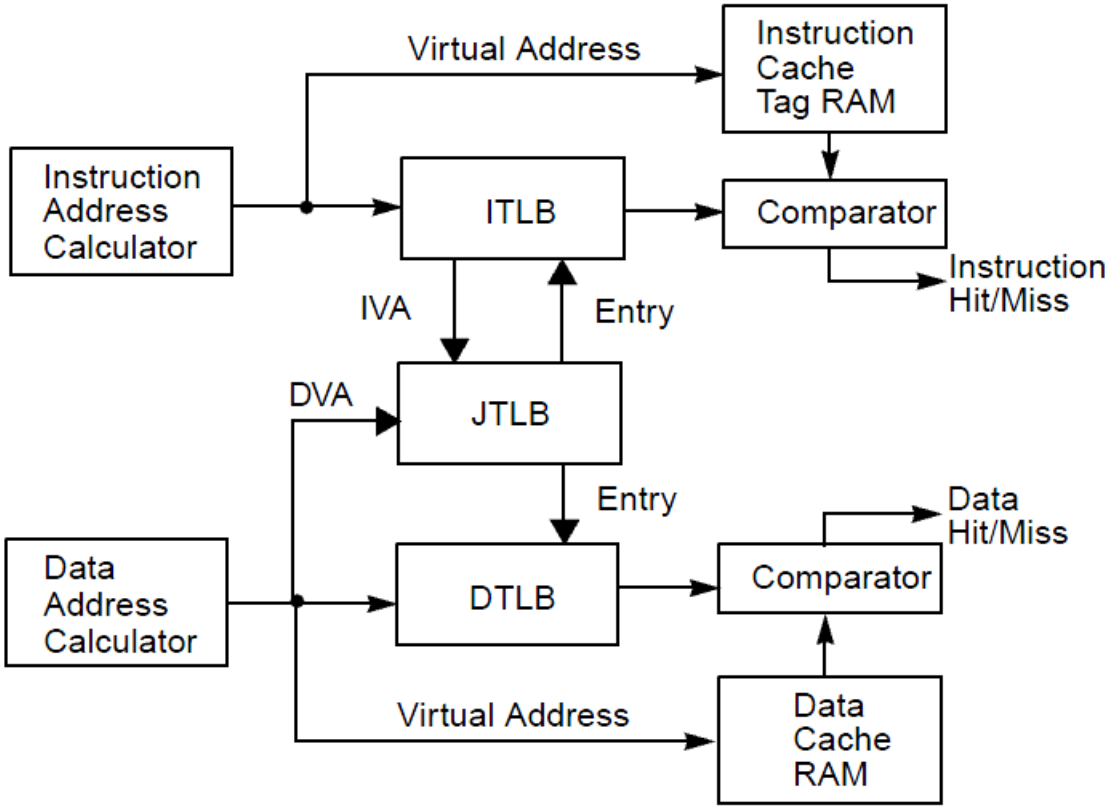
Хардверная реализация MMU в MIPS microAptiv UP

- Вместо одной ассоциативной памяти MMU использует три
  - ITLB, DTLB, JTLB
- Это трюк для повышения производительности
  - Дополнительный уровень кэширования
- Данная структура невидима для программиста
  - Для программы все выглядит как один TLB



Параметры реализации MMU в MIPS microAptiv UP

- TLB инструкций
  - Instruction TLB (ITLB)
  - 4 записи
  - Издержка промаха 2 цикла
- TLB данных
  - Data TLB (DTLB)
  - 4 записи
  - Издержка промаха 1 цикл
- Совмещенный TLB
  - Joint TLB (JTLB)
  - 16 записей



Регистры Cop0 для работы с TLB - 1

- **EntryHi – содержит VPN2 и ASID для чтения, записи и других операций**
- **PageMask – см. один из предыдущих слайдов**
- **EntryLo0, EntryLo1 – содержат PFN, V, RI, XI, D, C и G**
  - Бит G (Global) для записи ставится как G = EntryLo0.G and EntryLo1.G
- **PageGrain – содержит биты RIE и XIE разрешающие RI и XI в EntryLo**
  - При этом главное назначение Page Grain – поддержка страниц в 1KB (не в PIC32MZ)

Регистры Cop0 для работы с TLB - 2

- **Config1 – поле MMUSize содержит размер TLB минус единица**
  - Если MMUSize = 0, то TLB отсутствует и ядро использует FMT MMU (случай PIC32MX)
  - В PIC32MZ поле содержит число 15 (16 двойных записей TLB)

- **Index – индекс для чтения записи инструкцией TLBR и записи TLBWI**
  - Также служит для зондирования TLB инструкцией TLBP (TLB Probe)
  - TLBP записывает в регистр индекс найденной записи
    - Если не найдена – выставляет старший бит P (Probe Failure)

Что делать, если нужно много трансляций?

- Для многих систем количество трансляций в TLB недостаточно
- В таком случае таблица трансляций хранится в памяти, а TLB используется как кэш трансляций
- Если процессор не находит трансляцию в TLB, происходит исключение TLB Refill
- Обработчик исключения загружает трансляцию из памяти и записывает ее в случайную запись TLB

Структура таблицы страниц в памяти

- Таблица трансляций в памяти представляет собой линейный массив структур, состоящих из значений регистров Cop0 EntryLo0 и EntryLo1
- Структуры выровнены на границу 16 байт для совместимости с MIPS64

Адрес	Содержимое
PTEBase + 0	EntryLo0 для страницы 0
PTEBase + 4	
PTEBase + 8	EntryLo1 для страницы 1
PTEBase + 12	
PTEBase + 16	EntryLo0 для страницы 2
PTEBase + 20	
PTEBase + 24	EntryLo1 для страницы 3

Регистр Cop0 Context

PTEBase	BadVPN2	0

- **PTEBase [31:23] – Page Table Entry Base**
  - Записывается пользователем (операционной системой)
  - Определяет базовый адрес таблицы страниц в памяти
    - Весь регистр Context становится виртуальным адресом таблицы страниц, когда BadVPN2 = 0
- **BadVPN2 [22:4]**
  - Записывается процессором во время исключений TLB Refill, TLB Invalid и других
  - Соответствует битам [31:13] виртуального адреса, вызвавшего исключение
  - Во время исключения весь регистр Context образует адрес записи в таблице страниц
    - Context = { PTEBase, BadVPN2, 4 бита нулей для выравнивания }

Регистры Cop0 Random, Wired и команда TLBWR

- **Random**
  - Содержит псевдослучайное число в пределах от Wired до Config1.MMUSize
  - Используется командой TLBWR, TLB Write Random

- **Wired**
  - Содержит нижнюю границу для значения Random
  - Необходим, чтобы при использовании TLB как кэша трансляций из памяти, некоторые записи не выталкивались бы из TLB

- **Команда TLBWR**
  - Записывает содержимое регистров EntryHi, PageMask, EntryLo0 и EntryLo1 в запись TLB, индексом которой в TLB является регистр Random

Минимальный обработчик исключения TLB Refill

```
    .set    noreorder           // Выключить слишком умную оптимизацию
                                // ассемблером
tlb_miss_32:
    mfc0    k1,  C0_Context     // Взять адрес записи в таблице страниц
    lw      k0,  0(k1)          // Загрузить EntryLo0 из таблицы в памяти
    lw      k1,  8(k1)          // Загрузить EntryLo1 из таблицы в памяти
    mtc0    k0,  C0_EntryLo0
    mtc0    k1,  C0_EntryLo1
    ehb                         // Гарантия, что в регистры записалось
                                // перед использованием

    tlbwr                       // Записать EntryHi, PageMask,
                                // EntryLo0, EntryLo1 в случайное место TLB
                                // При TLB Refill, EntryHi уже установлен
    eret                        // Вернуться из исключения
```

Как экономить на размере таблицы страниц

- **Таблица страниц для 32-битного адресного пространства – 8MB**
  - Для 64-битного вообще необозримая
- **Для сокращения размеров саму таблицу можно хранить в виртуальной памяти, отображаемой с помощью TLB**
  - Context.PTEBase не обязан указывать на неотображаемой адрес
- **Когда страницы нет в TLB во время TLB Refill и вообще любого исключения (Cop0 Status.EXL=1), происходит другое исключение TLB Invalid на общий вектор исключений (не вектор для TLB Refill)**
  - Второе прерывание позволяет подновить таблицу страниц в памяти
  - При этом второе исключение возвращается прямо в код, вызвавший первое исключение
  - Исключение TLB Refill происходит снова, но на этот раз с обновленной таблицей и без последующего TLB Invalid

Другой пример реализации: MIPS32 Linux

- **Двухуровневая таблица страниц в памяти**

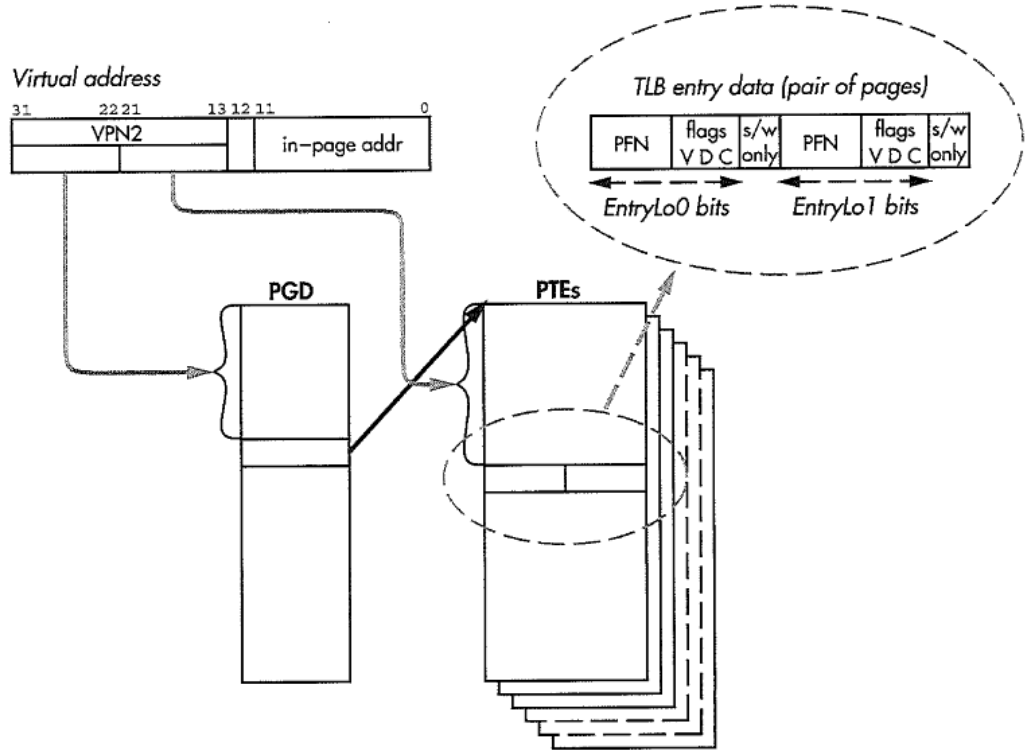


FIGURE 14.4 Linux two-level page table (32-bit MIPS design).

Источник:

Dominic Sweetman.

See MIPS Run Linux

Инициализация TLB

Необходима в boot code перед любой работой с TLB

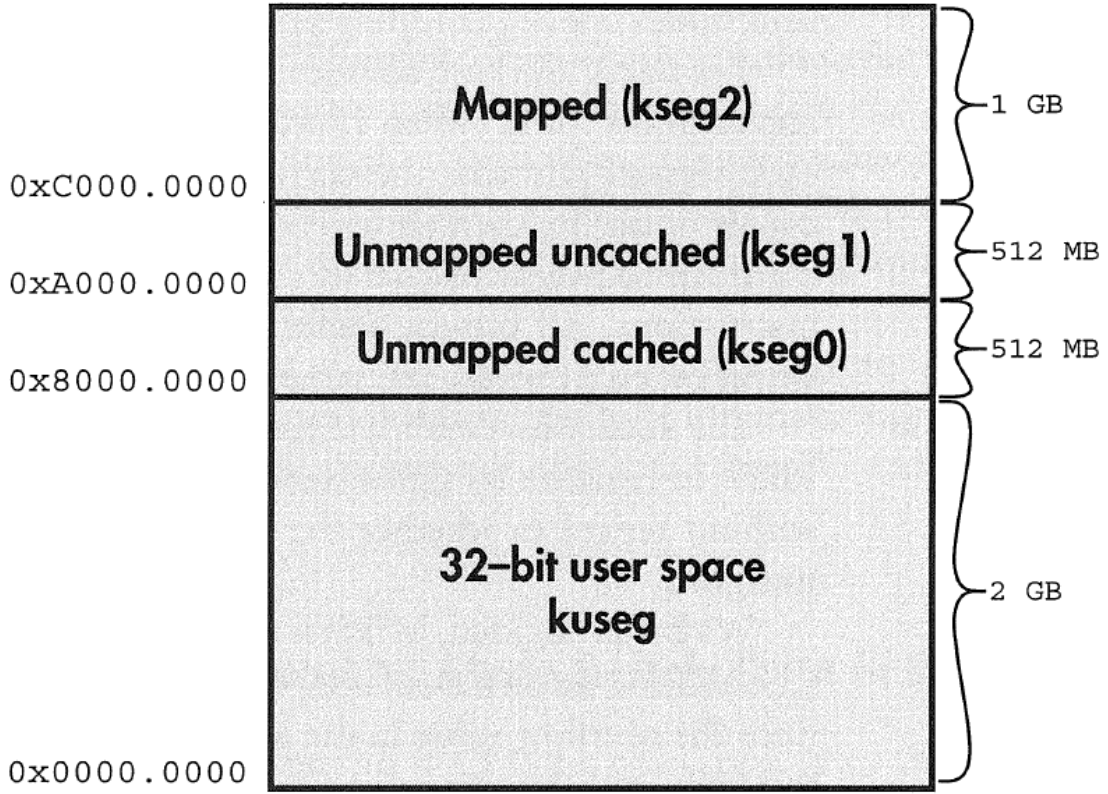
- **Инициализировать Context.PTEBase, Wired, PageGrain.RIE и PageGrain.XIE**
- **Записать нули в EntryHi, PageMask, EntryLo0 и EntryLo1**
- **Пройти в цикле все индексы TLB от нуля до Config1.MMUSize**
- **В теле цикла записывать Index, после чего выполнять инструкции EHB и TLBWI**
  - Инструкция EHB (Clear Hazard Barrier) нужна для гарантии, что запись в Index произошла перед выполнением TLBWI, которая записывает в TLB данные из EntryHi, PageMask, EntryLo0 и EntryLo1
- **Код, инициализирующий TLB, должен исполняться из области памяти, не использующий отображение с помощью TLB (kseg0 или kseg1)**

For Distribution

Карты виртуальных адресов в архитектуре MIPS

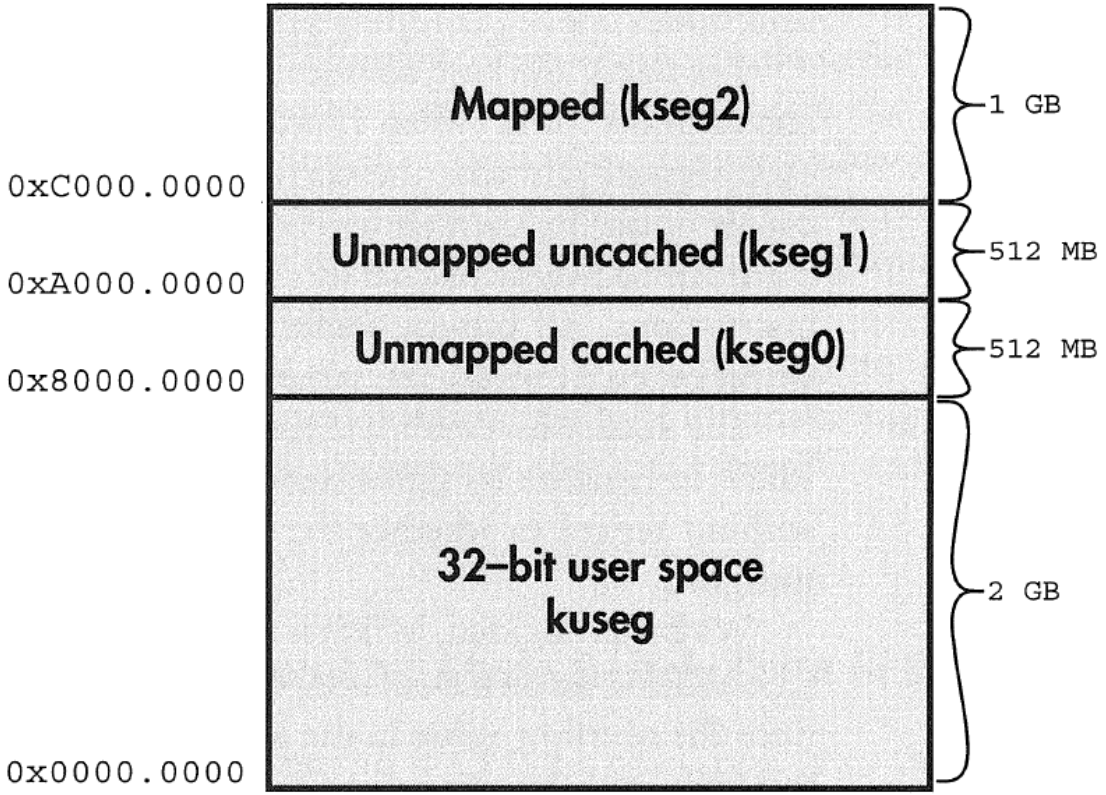
Классическая карта виртуальных адресов MIPS

- **Не все адреса транслируются TLB**
- **Сегменты kseg0 и kseg1**
  - Транслируются простым удалением верхних двух битов адреса
- **kseg0**
  - Кэшируемый
  - Используется для OS
- **kseg1**
  - Некэшируемый
  - Используется для загрузки – адрес 0xbfc00000
  - Используется для регистров ввода-вывода



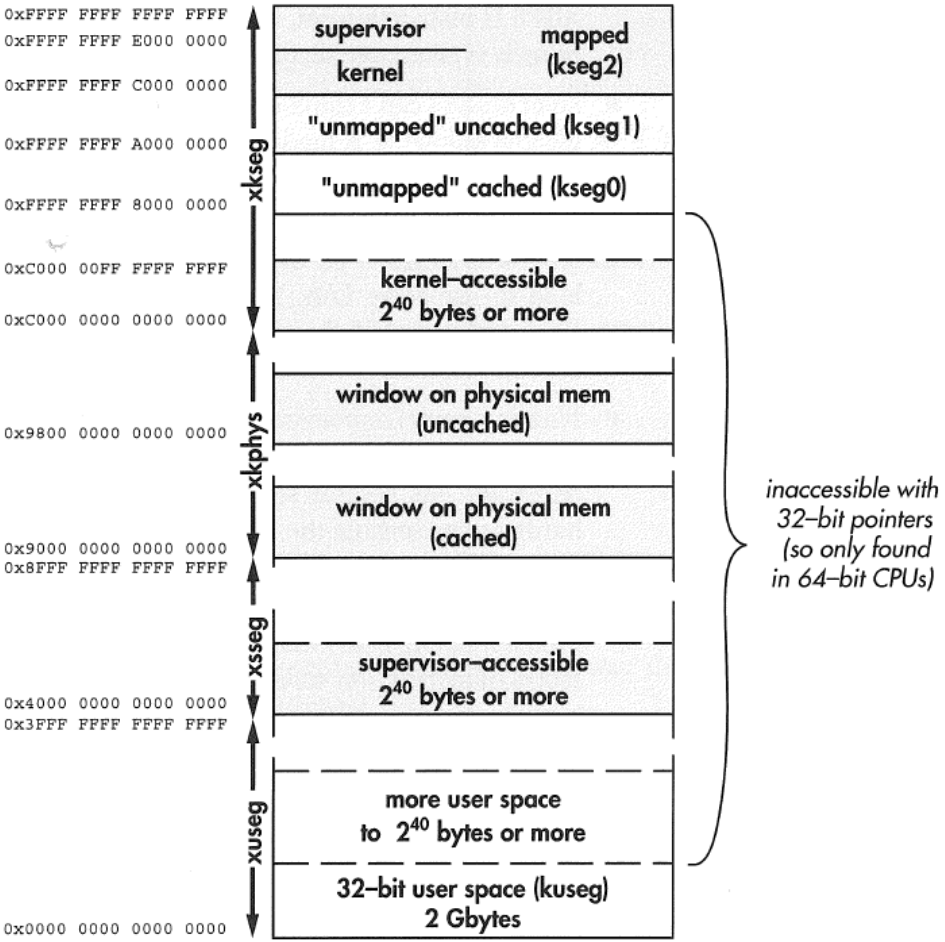
Классическая карта виртуальных адресов MIPS - 2

- **Пользователь использует только нижние два гигабайта - kuseg**
  - Отображаемые с TLB
  - В ядрах MIPS interAptiv и proAptiv пользователь может иметь доступ к 3.5GB с помощью нового механизма EVA (Enhanced Virtual Addressing)



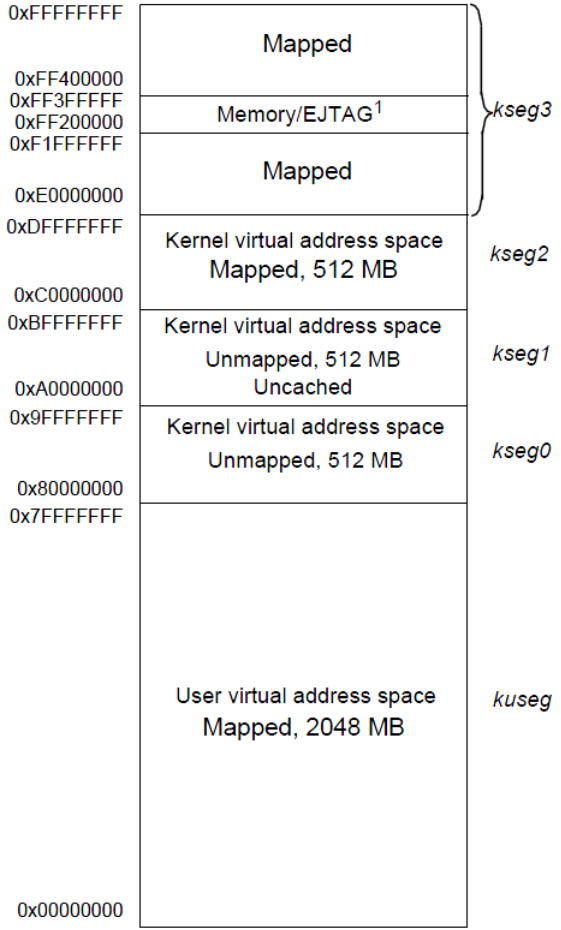
Для сравнения – карта виртуальных адресов MIPS64

- Если отрезать 32 верхних бита адреса – совместима с MIPS32
- Регион от 0x8000_0000 до 0xFFFF_FFFF_8000_0000 доступен только с 64-битными указателями
  - 64-битная карта «упакована» внутри 32-битной карты



Карта виртуальных адресов MIPS M4K и MIPS microAptiv UP

- Дополнительные детали – адреса для отладочного интерфейса EJTAG
- Для MIPS M4K TLB не используется, вместо этого используется FMT
  - Fixed Mapping Translation



For Distribution

Трансляция с фиксированным отображением
Fixed Mapping Translation (FMT)

Реализация MMU с помощью FMT

Fixed Mapping Translation – фиксированное отображение адресов

- **Доступные пользователю адреса в user-mode сдвигаются на 0x40000000**
  - Это способ имитации части поведения TLB без TLB как такового
- **Не защищает различные пользовательские программы друг от друга**
- **Требует минимального количества аппаратных ресурсов**
- **Опция FMT реализована на всех ядрах MIPS Technologies**
- **Используется в ядре MIPS M4K / микроконтроллере PIC32MX**

Трансляция виртуальных адресов в физические с помощью Fixed MappingTranslation

Figure 5 FMT Memory Map (ERL=0) in the M4K Core

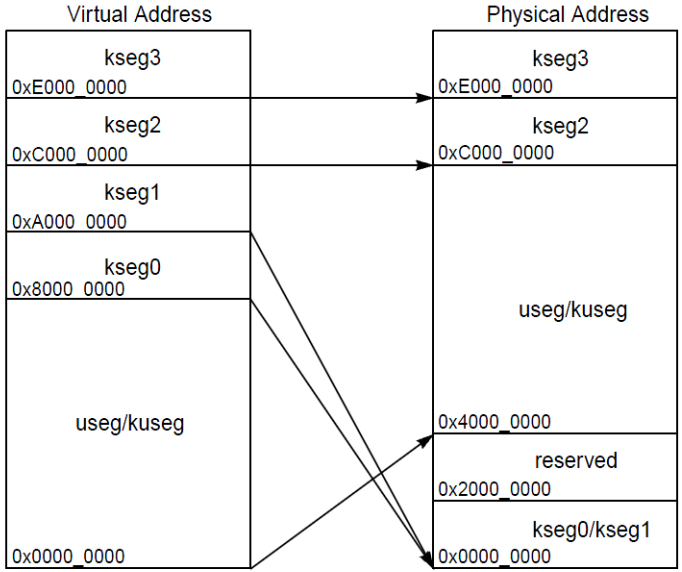
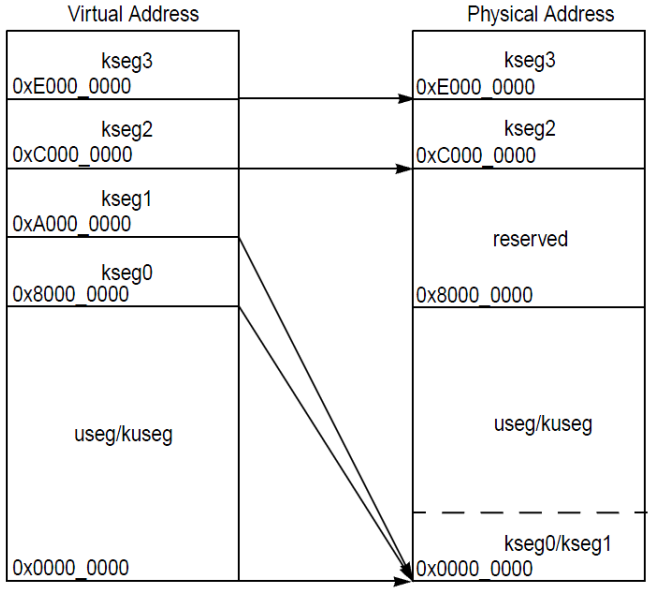


Figure 6 FMT Memory Map (ERL=1) in the M4K Core



Источник: MIPS32® M4K® Processor Core Datasheet March 4

For Distribution

Матрица шины PIC32MX, Bus Matrix (BMX)

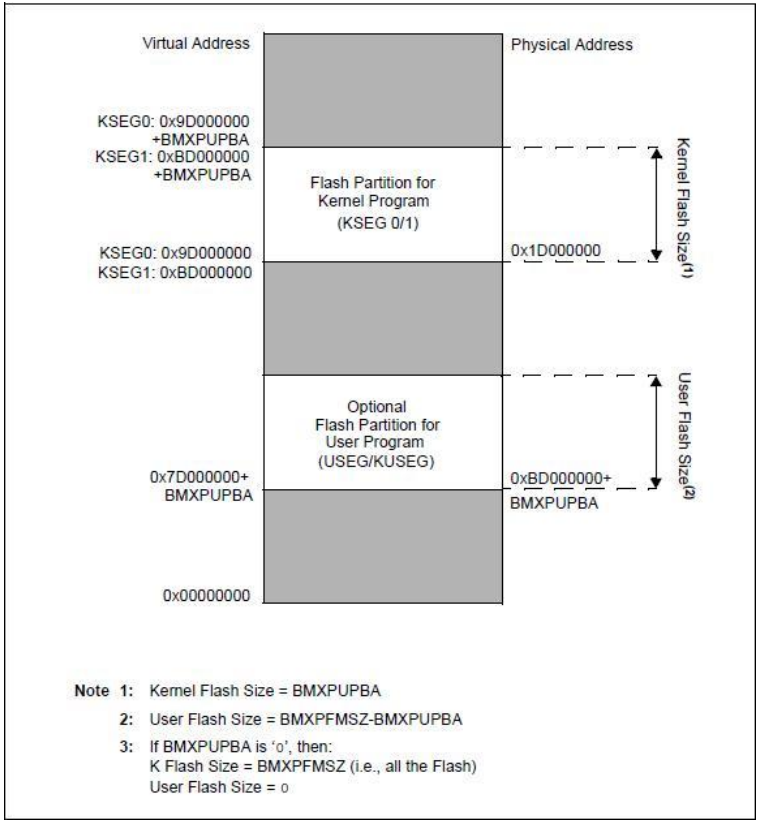
Матрица шины PIC32MX

Bus Matrix (BMX)

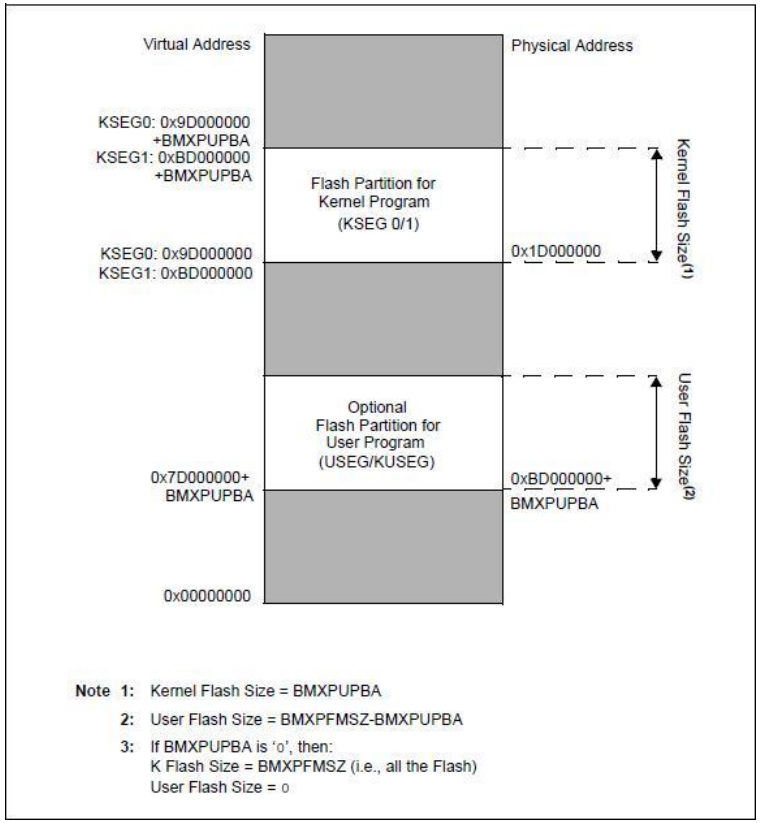
- **Дополнительный уровень трансляции физических адресов**
  - Задается доступными программисту регистрами микроконтроллера
  - Необходим, чтобы распределить физическую память после FMT-трансляции между областями, доступными только в привилегированном режиме (kernel-mode) и пользовательском режиме (user-mode)
- **BMX также определяет приоритеты различных устройств про доступе к памяти**
  - CPU Instruction Access
  - CPU Data Access
  - DMA
  - ICD / Debug
  - Initiator Expansion
- **И некоторые другие параметры**
  - Кешируемость DMA доступов к флэшу и т.д. – см. описание BMXCON в документации

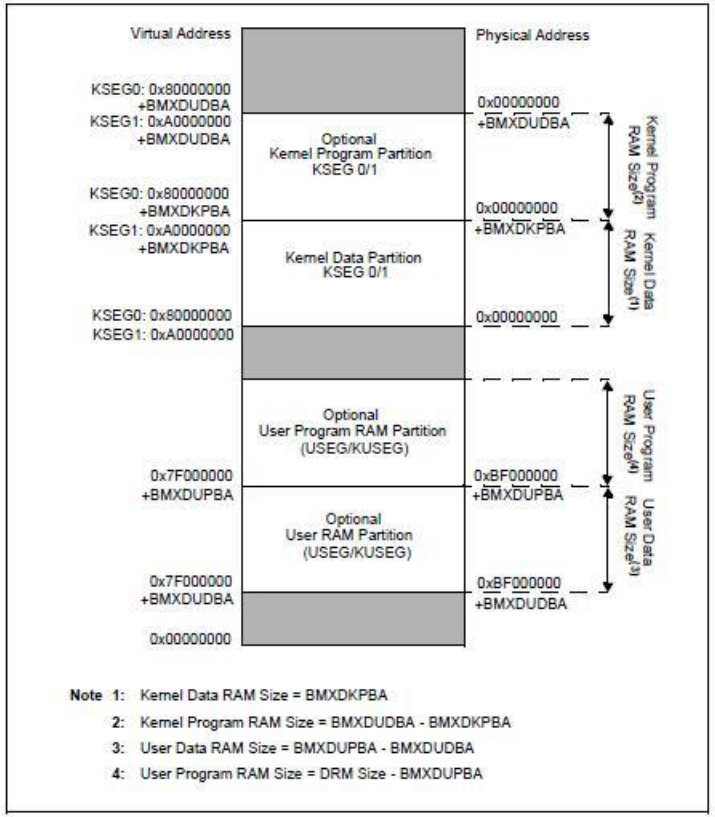
Разделение флэша между пользователем и OS

- Полный объем используемого флеша хранится в регистре BMXPFMSZ
- Разделение определяется регистром BMXPUPBA
  - Если BMXPUPBA = 0
    - Флеш для пользовательского режима не выделяется
  - Если BMXPUPBA = 1
    - Размер BMXPFMSZ - BMXPUPBA



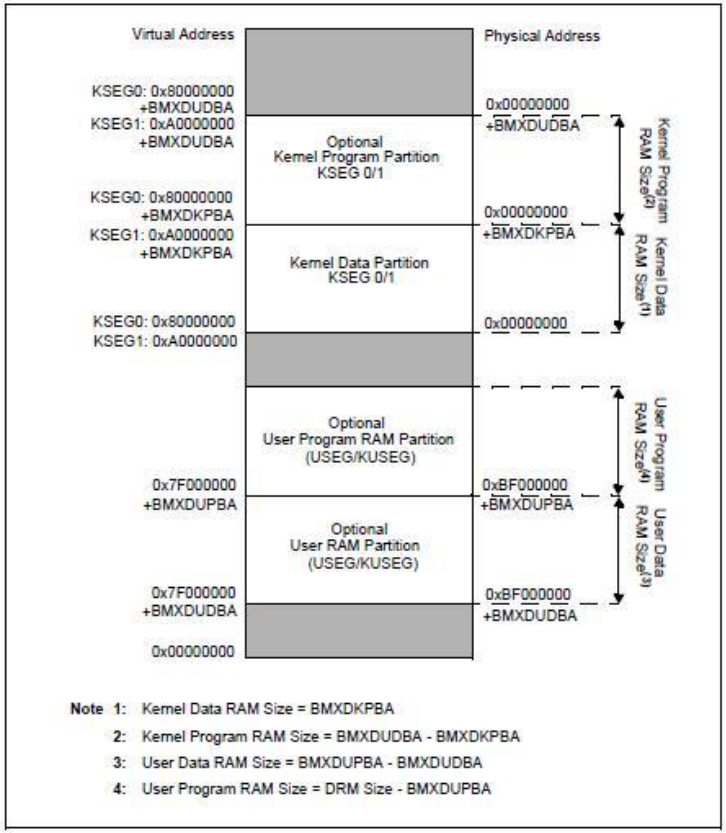
Разделение флэша и RAM





Разделение RAM между пользователем и OS

- Полный объем используемой RAM хранится в регистре BMXDRMSZ
- Если BMXDKPBA=0 или BMXDUDBA=0 или BMXDUPBA=0
  - Вся RAM выделяется для данных ядра
- Если все регистры выше не нулевые
  - Размер данных ядра = BMXDKPBA
  - Размер программы ядра = BMXDUDBA – BMXDKPBA
  - Размер данных пользователя = BMXDUPBA – BMXDUDBA
  - Размер программы пользователя = BMXDRMSZ - BMXDUPBA

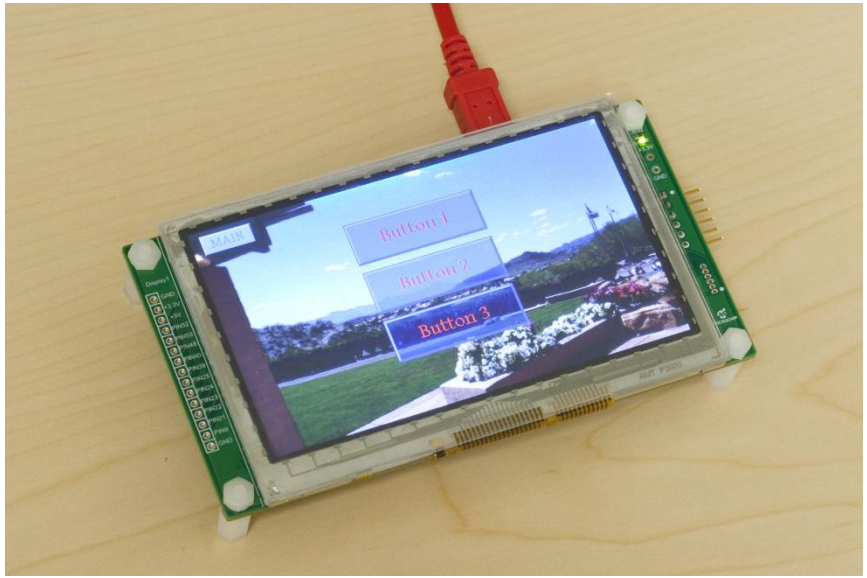


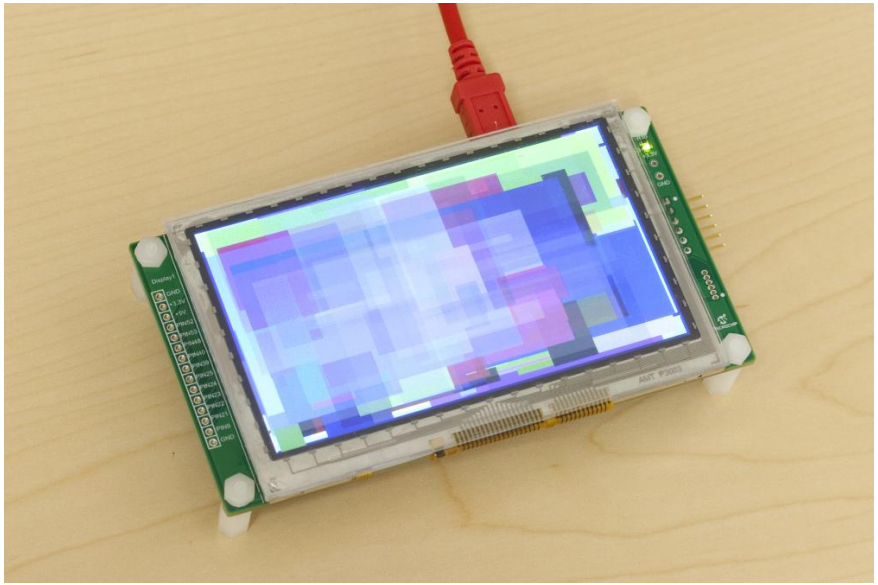
For Distribution

Контроллер прямого доступа PIC32MX, пример использования для графики

Демо: PIC32 может генерировать развертку графического дисплея без контроллера

- Вы видите устройство с графическим дисплеем TFT (Thin Film Transistor)
- У этого устройства нет контроллера
- Как оно работает?





Расчет количества циклов для генерации развертки удивляет

- Дисплей WQVGA – 480 x 272
- Каждый пиксел – 24-битный RGB – 3 байта
- Частота кадров в дисплее – 60 Hz
- Вывод каждого байта через параллельный порт – 3 цикла
- Следовательно, на генерацию развертки требуется не менее 480 * 272 * 3 * 60 * 3 = 70,502,400 циклов в секунду
- Но ведь максимальная частота используемого PIC32 – 80 MHz ?!
- Неужели процессор занимается ТОЛЬКО генерацией развертки ?

Разгадка – развертку генерирует DMA контроллер

- DMA контроллер в PIC32 – устройство, которое умеет пересылать информацию между периферийными устройствами и / или внутренней памятью без участия CPU
- DMA контроллер может на 95% разгрузить CPU от задачи генерации развертки
- Идея: Установить прерывание от таймера и DMA контроллера и в обработчике прерывания инициировать пересылку данных из памяти в параллельный порт с помощью DMA контроллера

Два способа использования – с внутренней или с внешней памятью

FIGURE 2: EXTERNAL MEMORY METHOD

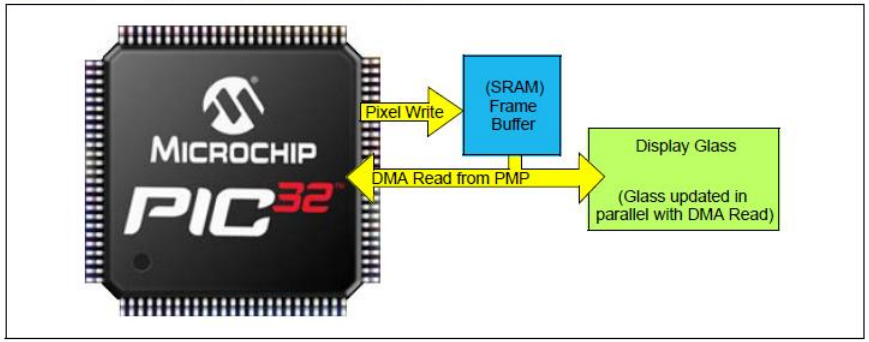
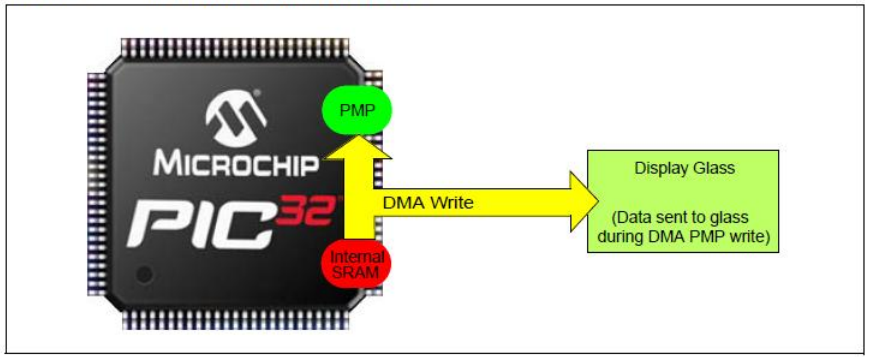


FIGURE 3: INTERNAL MEMORY METHOD



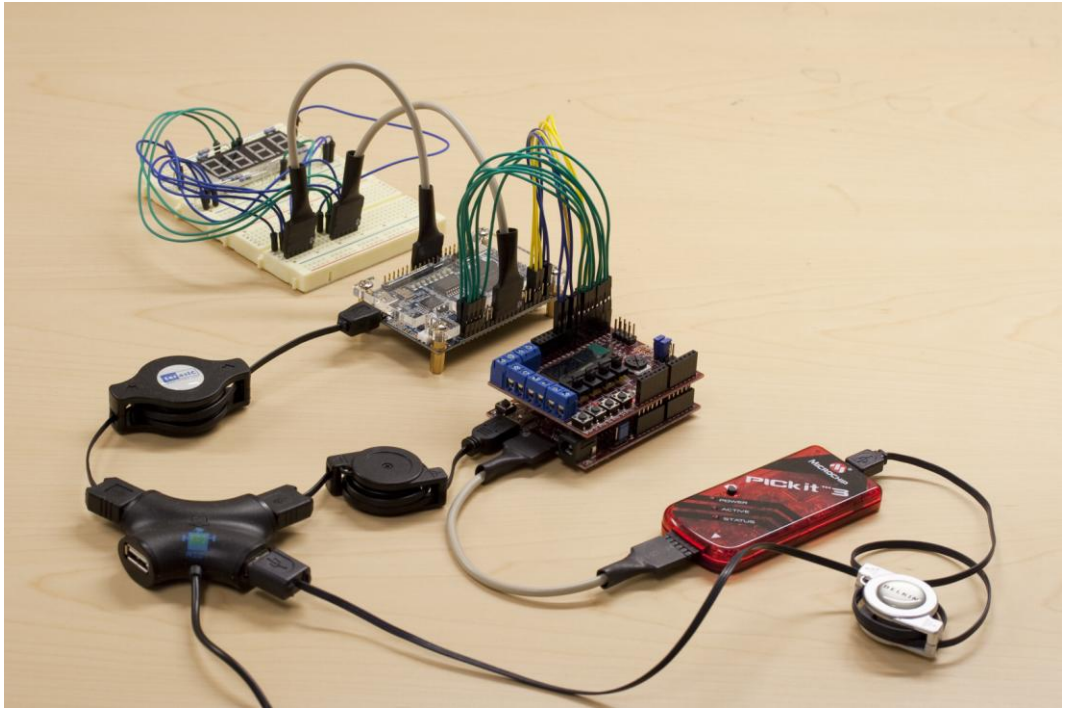
For Distribution

Соединение PIC32MX с ПЛИС / ППВМ / FPGA

Терминология

- **FPGA – Field Programmable Gate Array**
  - ППВМ – Программируемая Пользователем Вентильная Матрица
- **PLD – Programmable Logic Device**
  - Программируемая Логическая Интегральная Схема

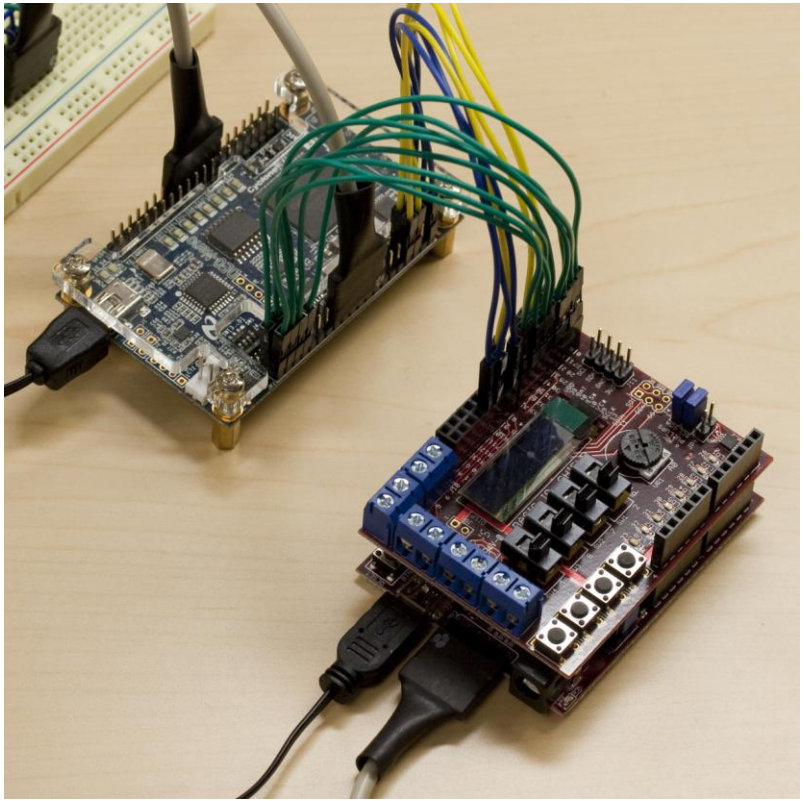
Демо: Интерфейс между микроконтроллером Microchip PIC32 и ПЛИС Altera Cyclone IV



Исходники демо - http://code.google.com/p/pic32-examples/source/browse/trunk/tedious/011_fpga_coprocessor_ports_3.X/

Демо крупнее: Интерфейс между микроконтроллером Microchip PIC32 и ПЛИС Altera Cyclone IV

- Плата Digilent chipKit Uno32 с Microchip PIC32
- Плата Digilent chipKIT Basic I/O Shield™
- Плата DE0-Nano с ПЛИС Altera Cyclone IV
- Отладчик PICkit 3
- Макетная плата с дисплеем, USB кабели

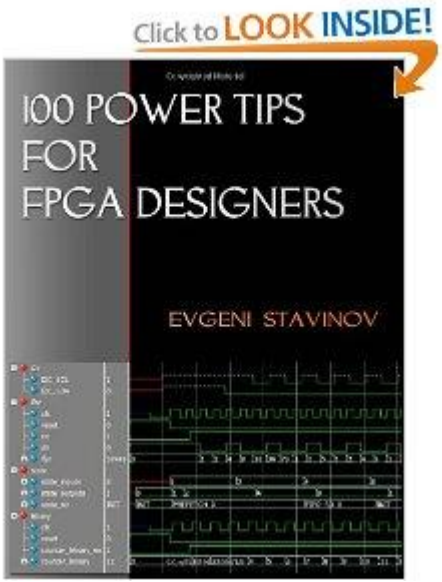


Детали интерфейса

- **Микроконтроллеры можно соединять с ПЛИС-ами либо просто через цифровые порты, либо через SPI, I^2C и другие протоколы**
  - Данное демо использует простые цифровые порты
- **Для трансфера данных из микроконтроллера в ПЛИС тактовая частота ПЛИС должна быть в несколько раз выше, чем у микроконтроллера - oversampling**
- **Так как тактовая частота в PIC32 и ПЛИС генерируется независимо, то нужно пропускать входящий в ПЛИС сигнал через два D-триггера (флип-флопа) чтобы (почти) избавиться от метастабильного состояния**
  - Метастабильное состояние при таком методе все же может появляться, но с очень низкой вероятностью - раз в несколько лет

Рекомендации по интерфейсу от Евгения Ставинова

- **Для приготовления этого демо я консультировался с экспертом по ППВМ Евгением Ставиновым**
- **Евгений работал в Xilinx и LeCroy, а также является автором книги 100 Power Tips for FPGA Designers**
- **http://outputlogic.com/**



For Distribution

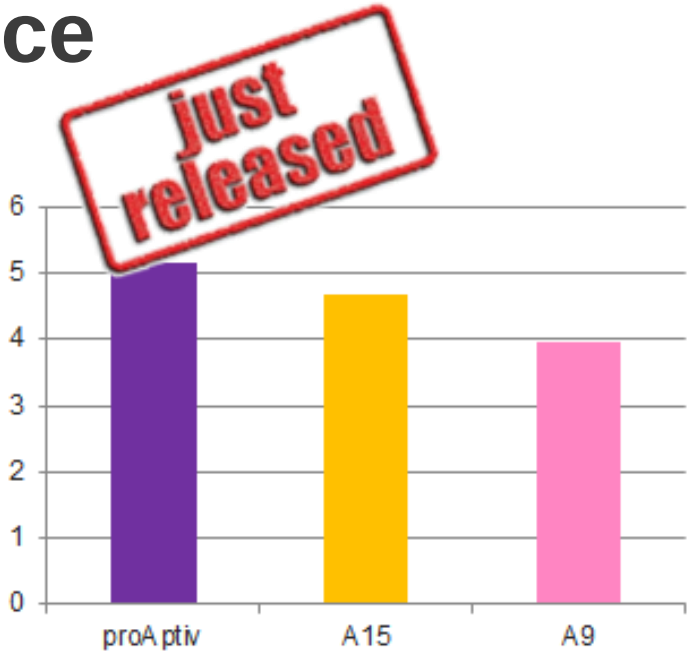
Сравнения MIPS и ARM

www.imgtec.com

Industry leading performance

- **Newly released CoreMark data**
  - 17th June 2013

- **CoreMark mimics modern workloads**
  - Data dependent branches, linked list, state-machine operations, integer arithmetic, etc
  - ~700,000 instructions vs. ~350 instructions for Dhrystone
  - Stresses CPU's branch prediction & L1 cache performance



IP	CoreMark/ MHz		IP	CoreMark/ MHz
proAptiv	5.10		Cortex A15	4.68
interAptiv	3.53		Cortex A9	3.97
microAptiv	3.44		Cortex M4	3.40
			Cortex M3	3.32

Built with GNU gcc head of tree (4.9) and using a gcc plug-in. Plug-in available at http://gcc.gnu.org/ml/gcc-patches/2013-05/msg00667.html

Преимущество MIPS M14K против ARM Cortex-M3

Согласно независимому аналитическому журналу

MICROPROCESSOR REPORT
www.MPRonline.com
THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

MicroMIPS Crams Code

New Processor Cores Introduce Denser 16/32-Bit Instruction Set

By Tom R. Halfhill (11/16/09-01)

Smaller is usually better for embedded processors, so MIPS Technologies is slimming down its 1980s-vintage instruction-set architecture. A new set of 16- and 32-bit instructions—dubbed microMIPS—uses less memory than existing 32-bit MIPS instructions and the 16-bit extensions added in the 1990s.

MicroMIPS will debut early next year in two new embedded-processor cores, the MIPS32 M14K and MIPS32 M14Kc. The M14K is an improvement on the MIPS32 M4K processor, introduced in 2002. It's a relatively simple, cacheless core intended for 32-bit microcontrollers in automobiles, industrial machinery, consumer electronics, and office equipment.

Its bigger brother, the M14Kc, is an improvement on the MIPS32 4KEc processor, introduced in 2003. The M14Kc has an MMU with a translation lookaside buffer (TLB), making it suitable for sophisticated embedded operating systems that manage virtual memory. It's designed for advanced consumer electronics, including DTVs, DVD players, set-top boxes, home networking equipment, personal entertainment devices, and digital cameras. Figure 1 shows how the M14K and M14Kc fit into the MIPS product line.

Both new processors respond much faster to interrupts and have better debugging features than the MIPS cores they supersede. Both gain advantages in clock speed, power consumption, and core size when compared with ARM's Cortex-M3 and ARM926 processors, and they give ARM's new Cortex-A5 a run for the money.

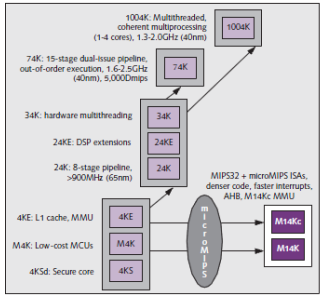


Figure 1. The new MIPS32 M14K and MIPS32 M14Kc processor cores introduce the microMIPS 16/32-bit instruction set and anchor the lower end of the MIPS product line. The M14K supersedes the M4K core, primarily for 32-bit microcontrollers. The M14Kc supersedes the 4KE core, offering an MMU for virtual-memory embedded operating systems.

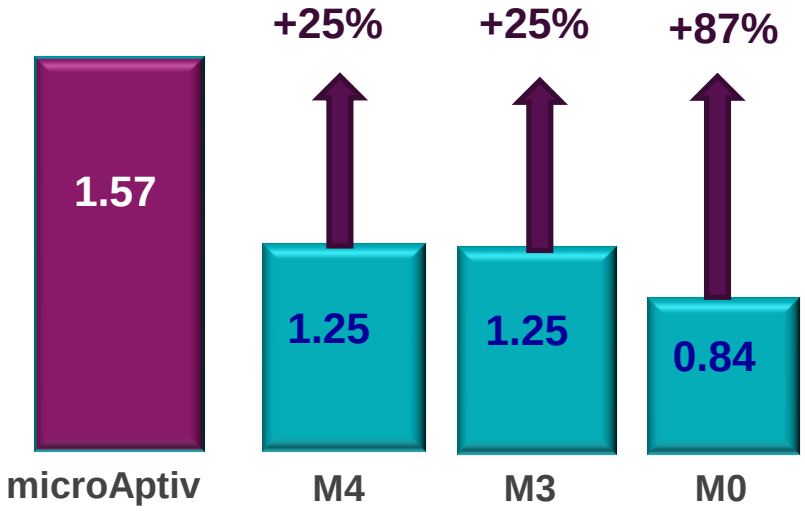
MIPS Fares Well Against ARM

Perhaps the most surprising result of comparing the new MIPS processors with ARM's best cores is that ARM no longer has a clear advantage in power consumption, core area, and performance. Usually, those are ARM's strengths.

For instance, using the data in Tables 2 and 3, we can compare the two microcontroller cores—the MIPS M14K and ARM Cortex-M3—in the same TSMC 90nm-G process. An area-optimized M14K will consume 11.6mW at 193MHz in 0.21mm^2 of silicon. A speed-optimized Cortex-M3 will consume 13.3mW at 191MHz in 0.37mm^2 of silicon. The M14K requires less power and silicon at virtually the same clock frequency. In power efficiency, the M14K wins, too: 25Dmips per milliwatt versus 17.9Dmips per milliwatt.

Note that we compared an area-optimized M14K with a speed-optimized Cortex-M3. That's because a speed-optimized M14K can reach a much higher clock frequency (295MHz). Assuming the two processors are clocked to deliver similar performance, the M14K will use less power and silicon. (The M14K has a throughput advantage of 1.5Dmips per megahertz versus 1.25Dmips per megahertz for the Cortex-M3.)

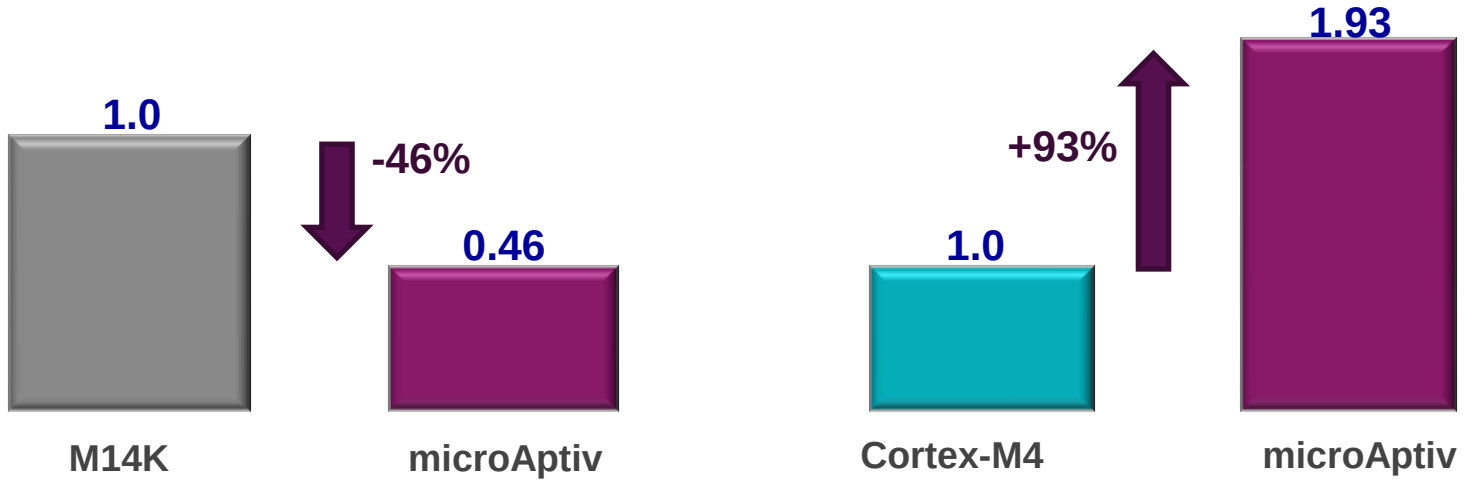
Better Performance vs ARM Cortex M Series



Dhrystone – DMIPS/MHz

- microAptiv outperforms Cortex M Series by up to 87%
- Lower clock speed needed to achieve equivalent performance

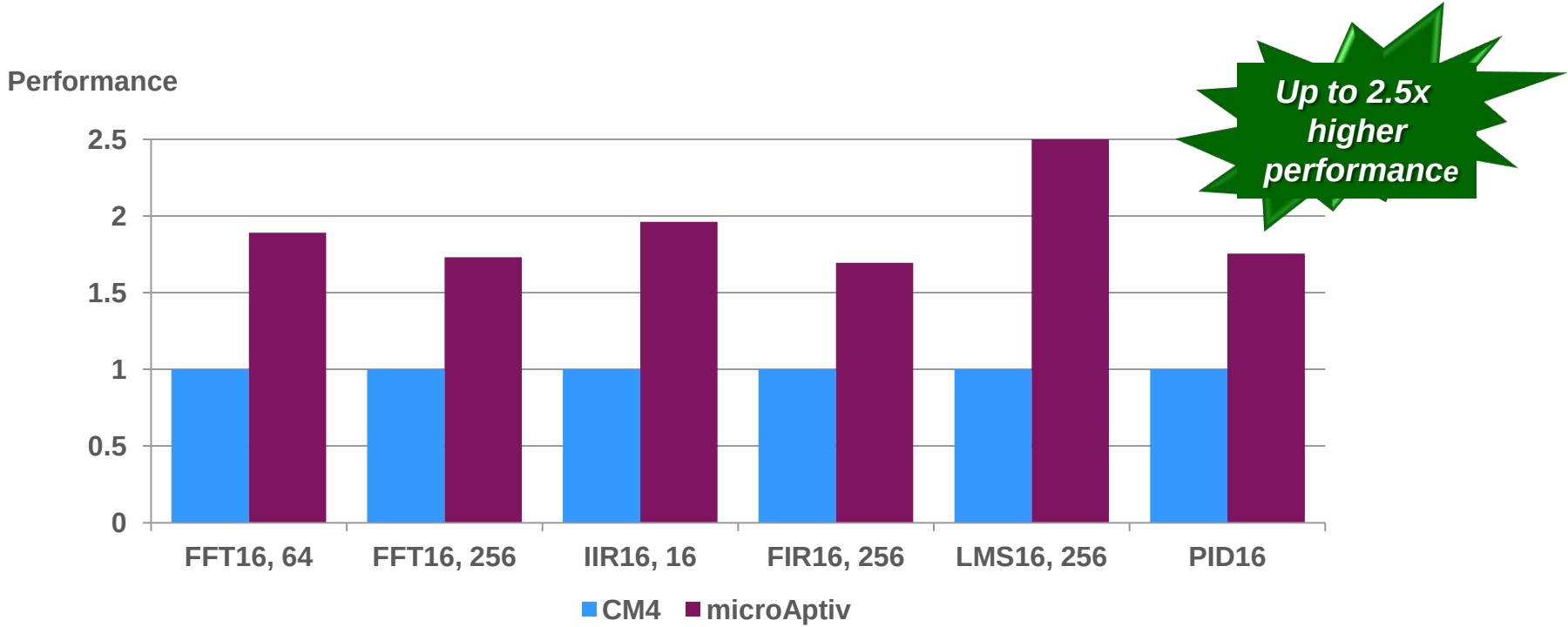
Better DSP Performance vs ARM Cortex M4



Running MIPS DSP Libs and ARM CMSIS DSP : FFT, FIR, IIR, LMS, PID, VMUL, H.264

microAptiv outperforms Cortex M4 by >90% for DSP operations

microAptiv vs. Cortex-M4 DSP Performance



Comparing DSP performance (higher is better) executing microAptiv DSP Library and Cortex M4 CMSIS Library

microAptiv vs Cortex-M Series Feature Comparison

Reduces Code Size With No Loss in Performance

Best Performance Efficiency

Better Real Time Performance

Faster Code Execution

More Flexible

FEATURE	microAptiv UC	Cortex-M3	Cortex-M0
Architecture	Harvard	Harvard	Von Neumann
Pipeline stages	5	3	3
ISA	MIPS32 and/or microMIPS	Thumb-2	Thumb/Thumb-2 (subset)
Legacy 32-bit decoder	Y - MIPS32 optional	N	N
Total instructions	300+	155	56
DMIPS Performance	1.57 DMIPS/MHz	1.25 DMIPS/MHz	0.9 DMIPS/MHz
Performance/Area Efficiency (90LP)	40 CM/MHz/mm2	27 CM/MHz/mm2	28 CM/MHz/mm2
GPRs	32	16	13
GPR sets (max)	16	1	1
Interrupt control	Y - int & ext	Y - int NVIC	32
Priority levels	8	4	4
Interrupt latency	10 cycles	12 cycles	16 cycles
Tailchaining	Y	Y	Y
MMU	Y - FMT	N	N
MPU (optional)	Y - up to 16 regions	Y - up to 8 regions	N
Multiply-Divide unit	Y	Y	Multiply only
Atomic bit instructions	Y	Y	N
Instruction-only trace	Y	N	N
PC sampling	Y	N	N
Perfomance counter	Y	N	N
Fast Debug Channel	Y	N	N
2-wire Debug	Y	Y	Y
Secure Debug	Y	N	N
Local Code Ram (max)	4GB	1GB	None
Local Data Ram (max)	4GB	1GB	None
Parity	Optional	N	N
Fast SRAM interface	Y	N	N
Flash memory prefetch	Y	N	N
External interface	AHB-Lite	AHB-Lite	AHB-Lite
Co-Processor interface	Y	N	N
Custom Instruction support	Y	N	N

microAptiv vs Cortex M4 Feature Comparison

FEATURE	microAptiv	Cortex M4
Core	M14K	Cortex-M3
Cache version	Y	N
Pipeline Stages	5	3
ISA	MIPS32 and/or microMIPS	Thumb2
Total instructions	300	155
DMIPS Performance (DMIPS/MHz)	1.57	1.25
Performance Efficiency (CM/mm2)	2200	2000
GPRs	32	16
GPR sets (max)	16	1
Closely coupled memory support	Y	N
MPU Support	Y	Y
MMU support	Y	N
Interrupt latency	10 cycles	12 cycles
Tailchaining	Y	Y
Instruction-only trace	Y	N
PC sampling	Y	N
Perfomance counter	Y	N
Fast Debug Channel	Y	N
Single wire debug	2-wire cJTAG	Y
FPU option	N	Y

microAptiv vs Cortex M4 Feature Comparison

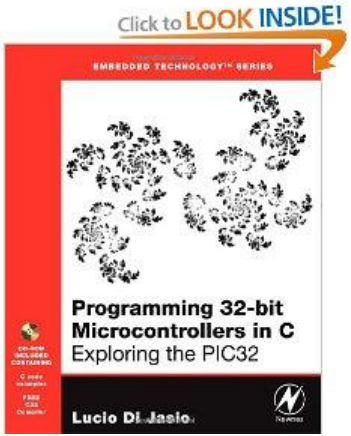
DSP FEATURE	microAptiv	Cortex-M4
DSP Instructions	159	80
SIMD - 8/16	Y	Y
SIMD Instructions	70	38
Integer & Fractional data types	Y	Y
Saturate, Rounding options	Y	Y
Dedicated DSP/MDU unit	Y	N
Accumulators	Y (4)	N
Multiply/MAC instructions	38	29
32x32, 16x16, dual 16x16	Y	Y
16x8	Y	N
dual 8x8	Y	N
32x16	N	Y
Single cycle instructions	Y	Y
Shift Instructions	Y	N
Bit Manipulation	Y	Y
Compare/Pick	Y	N
Data Pack/Unpack	Y	Y
Bit Reversed Addressing	Y (instruction)	Y (instruction)
Modulo addressing	Y (instruction)	N

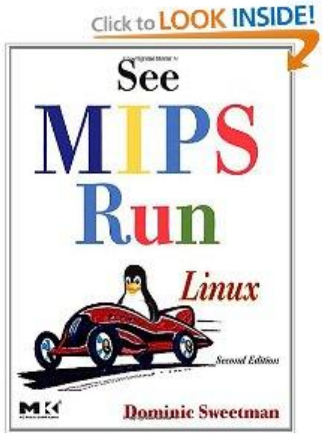
For Distribution

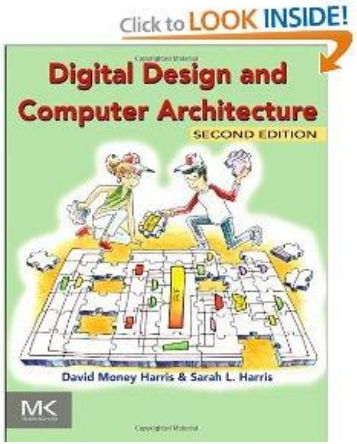
Рекомендуемая литература

Рекомендуемая литература по Microchip PIC32

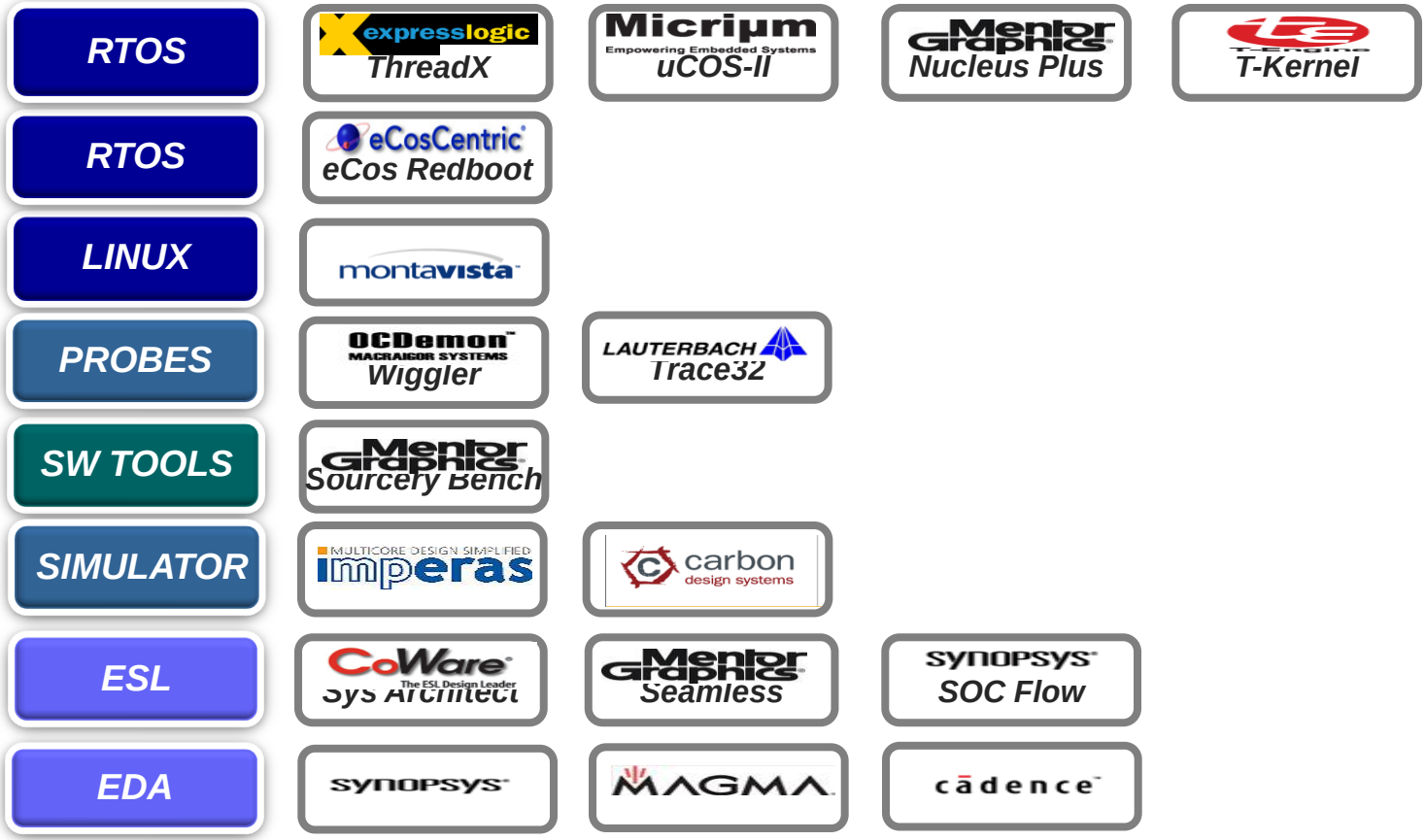
- Особо следует отметить David Harris and Sarah Harris, Digital Design and Computer Architecture для обучения студентов
  - В этой книге студенты строят подмножество MIPS-процессора с помощью Verilog и ППВМ / FPGA
  - После чего приводится пример индустриального MIPS – Microchip PIC32 и разбирается его периферия – порты, SPI, UART, аналог, соединение с моторами
  - Таким образом, теория соединяется с практикой, и hardware соединяется с software







microMIPS Ecosystem - current



For Distribution

Спасибо!