Experiment with Linux and ARM Thumb-2 ISA

Philippe Robin

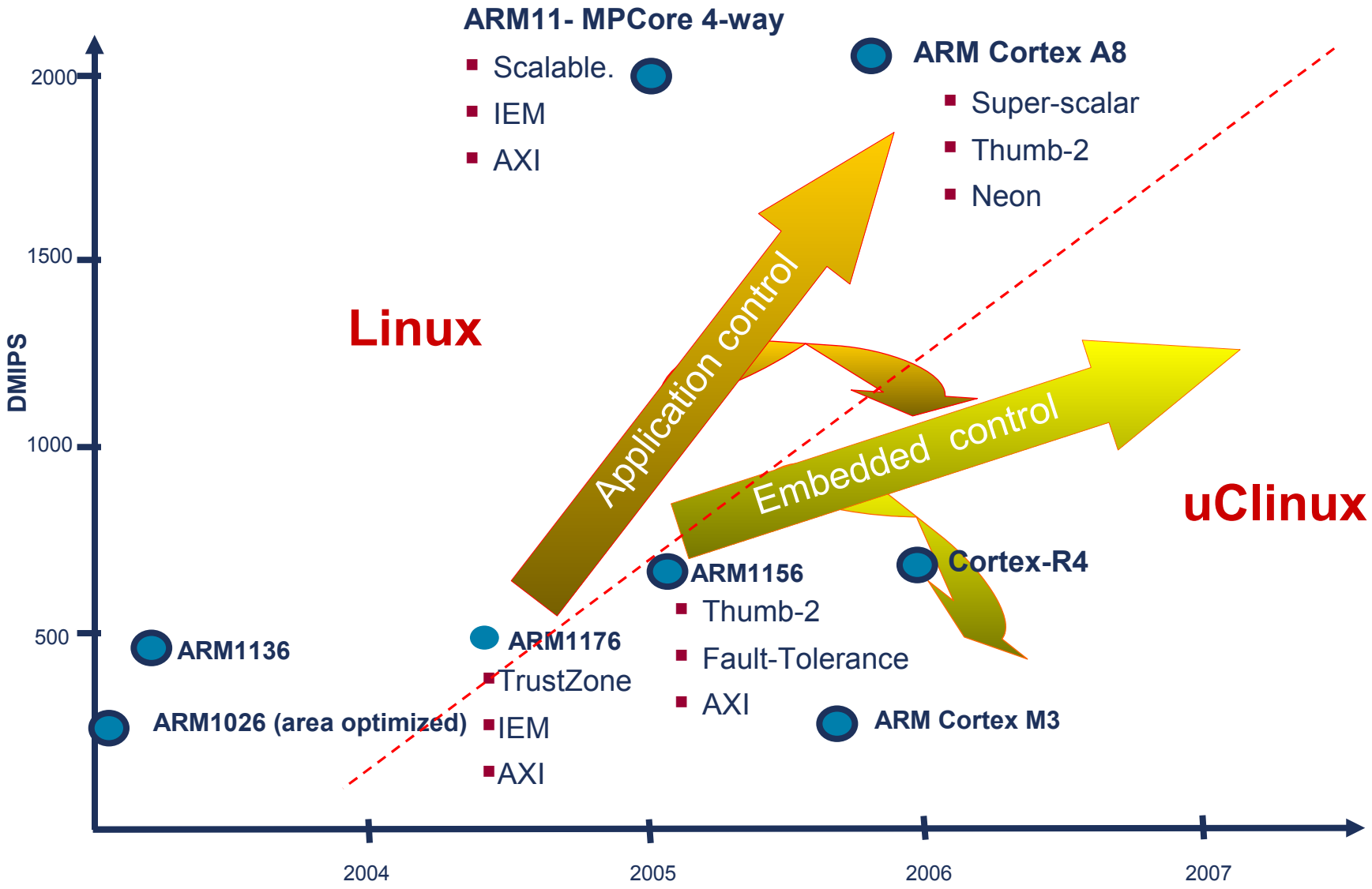
ARM Ltd.

Summary

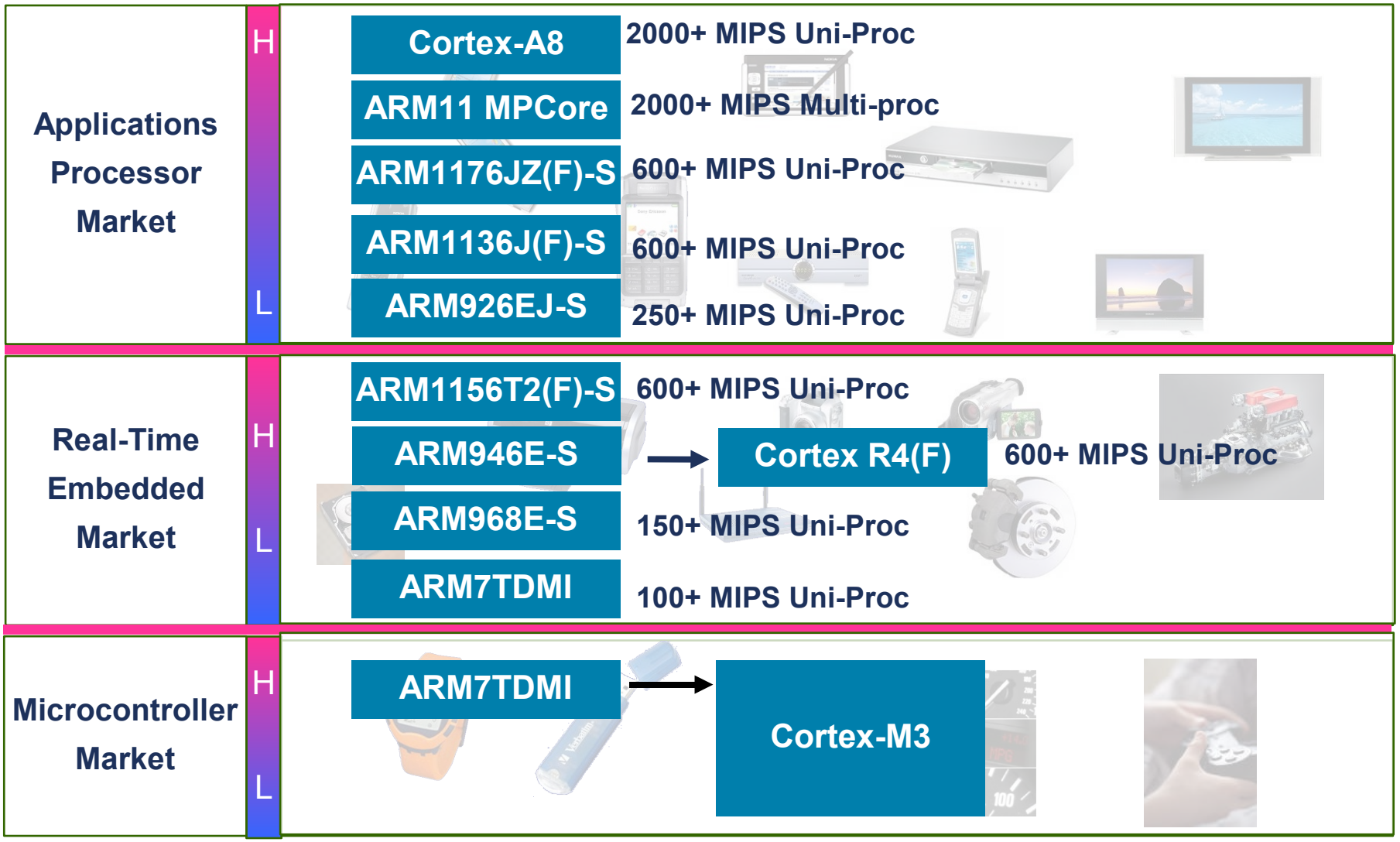
- ARM Roadmap and Processor Families
- Performance vs Code Size and ISA selection process
- Thumb-2 encoding and new instructions
- Changes in the Linux kernel
- Size reduction with kernel, libraries and applications
- Exception handler example
- Summary

ARM Activities

Linux and ARM Processor Roadmap

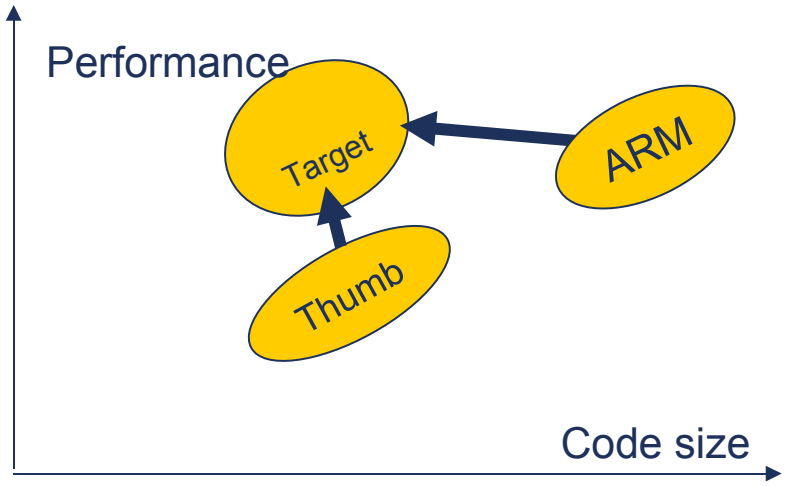


Processors Families



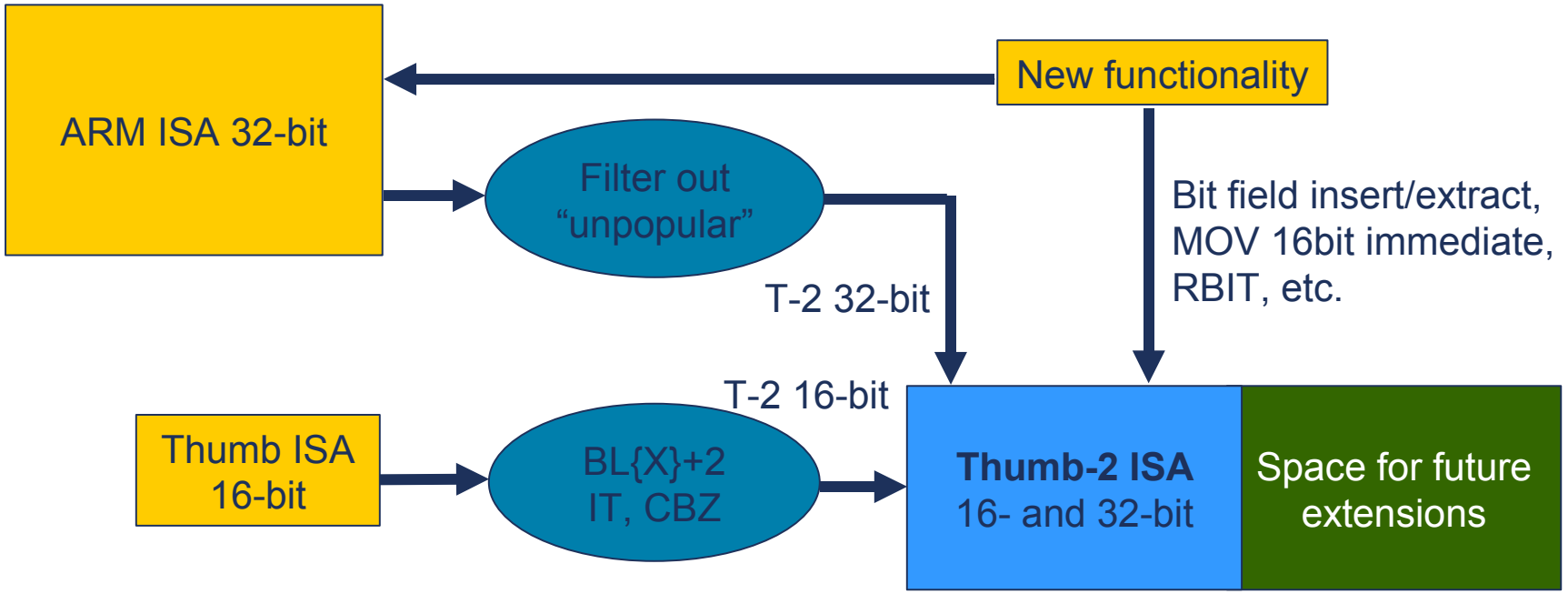
The Performance vs. Code Size Dilemma

- Thumb 16-bit ISA was created by analysing 32-bit ARM Instruction Set and deriving best fit 16-bit instruction set, thus reducing code size
  - User required to “blend” instruction sets by compiling performance critical code to ARM and the rest to Thumb
- But manual code blending is not optimal
  - Requires profiling
  - Modifications can reduce performance
  - Best results obtained near the end of the project
  - Difficult to manage distributed development



- A “blended ISA” is a better solution

The ISA Selection Process



- The ARM Thumb-2 core technology combines 16- and 32-bit instructions in a single instruction set and allows programmers / compilers to freely mix the instructions together without mode switching.

Thumb-2 Encoding

hw1[15:13]	hw1[12:11]	Length	Functionality
not 111	xx	16 bits	Current 16-bit Thumb instruction
111	00	16 bits	Current 16-bit Thumb B unconditional
111	not 00	32 bits	Thumb-2 32-bit

- Halfword pairs (hw1, hw2) of instructions are inserted into Thumb (thm) instruction stream.

The encodings selected are compatible with the existing Thumb BL and BLX instructions:

```
                        hw1                      hw2
Thumb BL{X}:    11110 offset[22:12]     111n1 offset[11:1]
T-2   BL{X}:    11110 offset[22:12]     11AnB offset[11:1]
```

- Two extra offset bits are generated by XORing the A and B bits with Offset[22]. This means that the offset is sign-extended when A = B = 1, which ensures backwards compatibility with the existing instructions.

Thumb-2 32-bit Instructions

- ARM-like
  - Data Processing Instructions
  - DSP and Media instructions
  - Load and Store instructions
  - Branch instructions
  - System control – BXJ, RFE, SRS etc.
  - Coprocessor (VFP, MOVE™, etc.)
- New
  - Bitfield insert/extract/clear BFI, {S|U}BFX, BFC
  - Bit reverse RBIT
  - 16 bit immediate instructions MOVW, MOVH
  - Table branch TB{B|H} [Rbase, Rindex]
  - Additional memory system hints (PLI)

Thumb-2 Move 16-bit Constant

Two 32 bit instructions to load a 32 bit constant, one instruction for each half word

- Replaces one 32 bit instruction and a 32 bit literal (ARM) or one 16 bit instruction and a 32 bit literal (Thumb)
  - Single MOVW would be used for the majority of cases
- Reduce the size of literal pools
- Reduce data access to I-TCM via D-side for constant loads (~5X)

```
MOVW Rd,#imm16
    Rd = ZeroExtend( imm16 )

MOVT    Rd,#imm16
   Rd[31:16] = imm16 // Rd[15:0] unaffected
```

Thumb-2 Bit Field Instructions

Allow insertion and extraction of signed/unsigned bit fields

- Provides better handling of packed structures
- Replaces bit mask and shift operations

BFC, BFI, SBFX, UBFX

ARM* or Thumb-2	ARM
BFI R0, R1, #bitpos, #fieldwidth	AND R2, R1, #bitmask BIC R0, R0, #bitmask << bitpos ORR R0, R0, R2, LSL #bitpos

Thumb-2 Table Branch Instructions

New Base + Offset Branching mechanism for switch statements generates branch targets directly from a table of destination offsets

- Thumb-2 code size as small or smaller than Thumb –Ospace
- Thumb-2 code performance as fast as ARM –Otime
- Thumb-2 code executes in a single instruction and uses packed table

New Thumb-2 Flow Control Instructions

Compare and Branch

CBZ Rn, <label>
CBNZ Rn, <label>

- Optimises for the common case of “Branch If Zero” or “Branch If Non-Zero”

ARM	Thumb-2	Thumb
CMP r0, #0	CBZ r0, ln	CMP r0, #0
BEQ ln		BEQ ln
8 Bytes, 1 or 2 cycles	**2 Bytes, 1 cycle**	**4 Bytes,1 or 2 cycles**

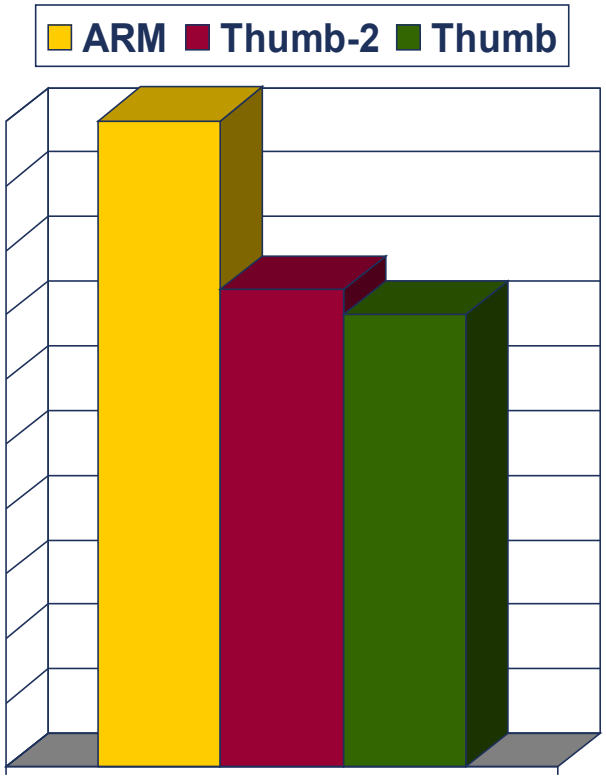
If-Then Conditional

IT{x{y{z}}} <cond>

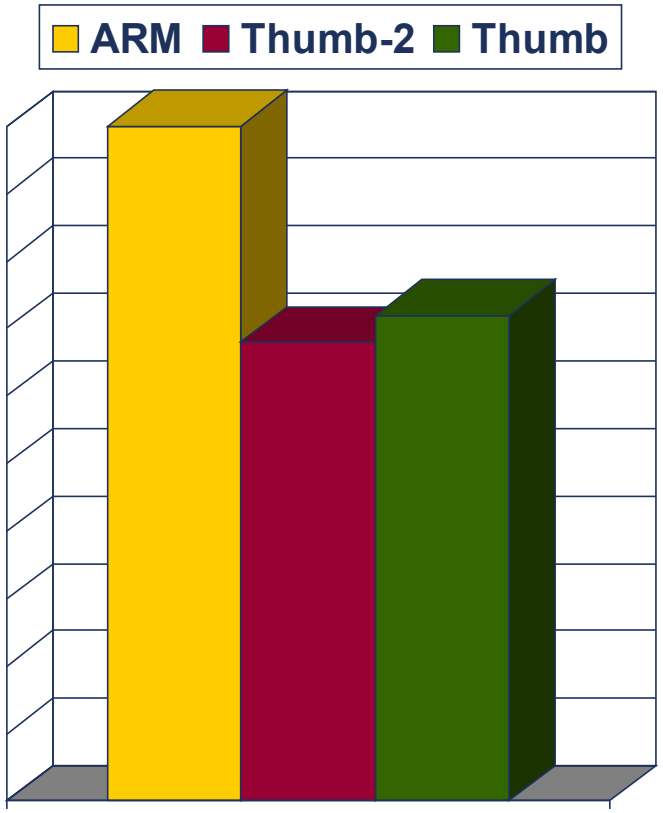
- The If-Then (IT) instruction causes the next 1-4 instructions in memory to be conditional
- Allows short conditional execution bursts in 16-bit instruction set

ARM	Thumb-2	Thumb
LDREQ r0, [r1]	ITETE EQ	BNE l1
LDRNE r0, [r2]	LDREQ r0, [r1]	LDR r0,[r1]
ADDEQ r0, r3, r0	LDRNE r0, [r2]	ADD r0, r3, r0
ADDNE r0, r4, r0	ADDEQ r0, r3, r0	B l2
	ADDNE r0, r4, r0	l1 LDR r0,[r1]
		ADD r0, r4, r0
		l2 ……
16 Bytes, 4 cycles	**10 Bytes, 4 or 5 cycles**	**12 Bytes, 4 to 20 cycles**

Thumb-2 Compiled Code Size



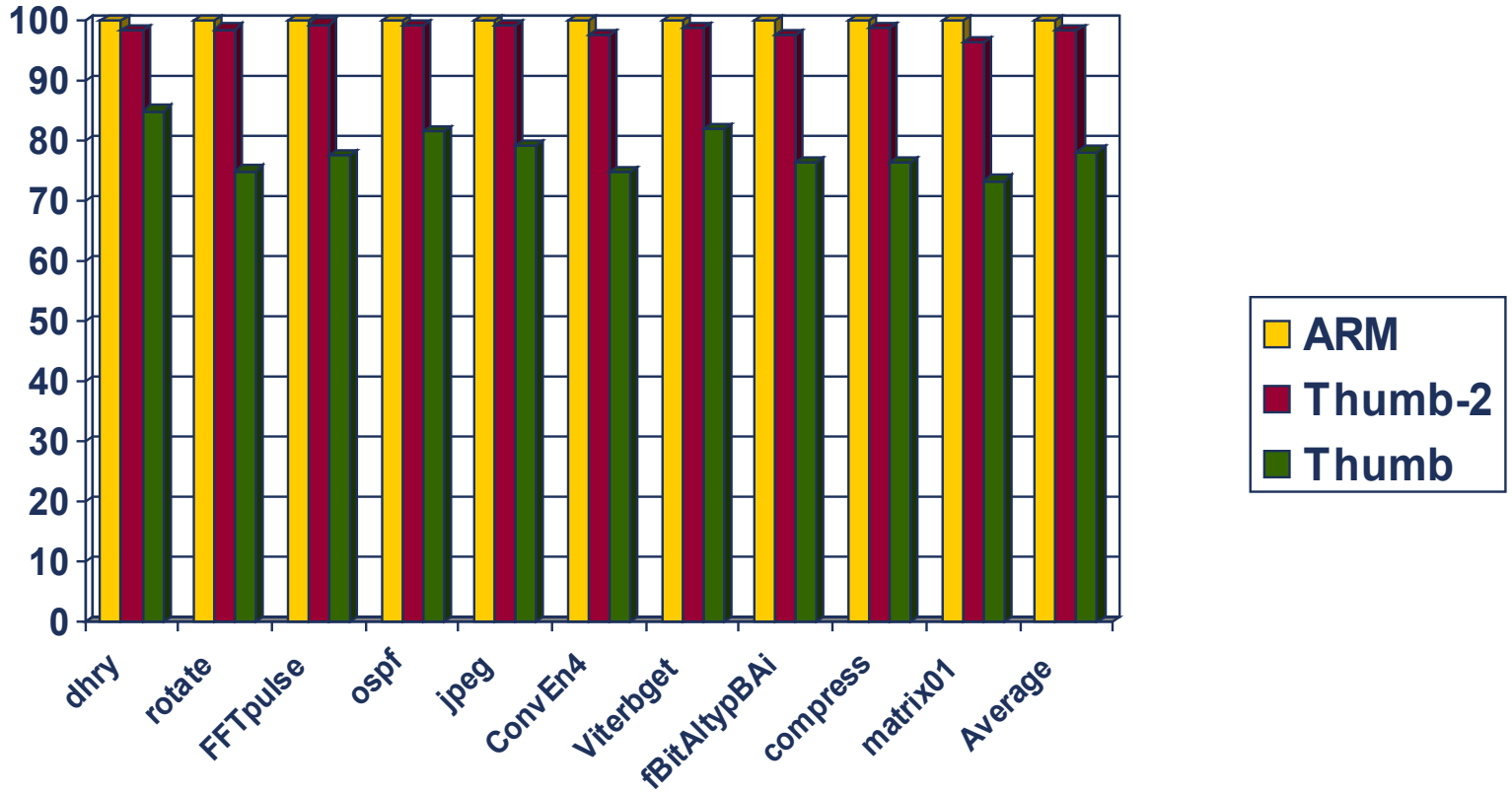
- Thumb-2 Performance Optimized
  26% smaller than ARM



- Thumb-2 Space Optimized
  32% smaller than ARM

Thumb-2 Performance

Analysis of the performance of code for EEMBC* benchmarks on ARM11 like cores



- Thumb-2 performance is 98% of ARM performance
- Thumb-2 code achieves 125% of Thumb performance

** Uncertified EEMBC benchmarks based information showing relative performance ONLY*

Thumb-2 – Changes to Linux Kernel

- A new control bit has been introduced with ARMv7 to control whether exceptions are taken in ARM or Thumb state
  - Modified Interrupt and Exception handling code accordingly
- Most 32-bit Thumb instructions are unconditional (whereas most of ARM instructions can be conditional)
- Many changes are due to adding unified syntax and flow control instructions
  - Use of *If-Then* (IT) instruction for instance
- There is no increase in the number of general purpose or special purpose registers, and no increase in register sizes
- Most Thumb 32-bit instructions cannot use the PC as a source or destination register.
- BL and BLX instructions are treated as 32-bit instructions instead of two 16-bit instructions
  - Note that 32-bit Thumb instructions can only take exceptions on their start address
- New T variants of LDR, STR
- New variants of LDREX and STREX
  - Thumb-2 has B, H, and D (Byte, Halfword, and Doubleword) variants

ARM vs Thumb-2 Memory Footprint

- Using GCC 4.1 with –O2 option
  - Average 20% size reduction on common libraries
  - Kernel is 29% smaller in Thumb-2 compared to ARM

	ARM Mode	Thumb-2 mode	Ratio
libc-2.3.6.so	1123552	824544	73%
libm-2.3.6.so	669496	542520	81%
2.6.19 kernel	1019832	724888	71%
MPlayer	5793064 (dynamic)	5619000 (dynamic)	96%
	6707792 (static)	5176036 (static)	77%

Sample - Exception Handler in Thumb-2

```
        .macro    vector_stub, name, mode, correction=0
vector_\name:
        .if \correction
        sub.w     lr, lr, #\correction
        .endif

        @ save lr_<exception> (parent PC) and spsr_<exception>
        @ (parent CPSR) to the SVC stack
        srsdb     sp, #SVC_MODE

        @
        @ Switch to SVC32 mode, save sp and lr and set up the stack.
        @ IRQs remain disabled.
        @
        mrs       lr, cpsr
        eor.w     lr, lr, #(\mode ^ SVC_MODE)
        msr       cpsr_cxsf, lr

        @ may be overwritten by the usr handlers
        str.w     sp, [sp, #(S_SP - S_FRAME_SIZE)]  @ save sp_svc to the SVC stack
        str.w     lr, [sp, #(S_LR - S_FRAME_SIZE)]  @ save lr_svc to the SVC stack

        sub.w     sp, sp, #S_FRAME_SIZE

        @
        @ the branch table must immediately follow this code
        @
        ldr.w     lr, [sp, #S_PSR]      @ read the saved spsr_<exception>
        and.w     lr, lr, #0x0f
        add.w     lr, pc, lr, lsl #2    @ address in the branch table
        ldr.w     pc, [lr, #4]          @ branch to handler in SVC mode
        .endm
[...]
        .macro    svc_entry
        stmia     sp, {r0 - r12}
        .endm
[...]
__dabt_svc:
        svc_entry

        add       r0, sp, #S_PC

        @
        @ get ready to re-enable interrupts if appropriate
        @
        mrs       r9, cpsr
        tst       r3, #PSR_I_BIT
        it        eq
        biceq     r9, r9, #PSR_I_BIT

        @
        @ Call the processor-specific abort handler:
        @
        @  r2 - aborted context pc
        @  r3 - aborted context cpsr
        @
        @ The abort handler must return the aborted address in r0, and
        @ the fault status register in r1.  r9 must be preserved.
        @
        ldmia     r0, {r2, r3}          @ load the lr_<exception> and spsr_<exception>
#ifdef MULTI_ABORT
        ldr       r4, .LCprocfns
        mov       lr, pc
        ldr       pc, [r4]
#else
        bl        CPU_ABORT_HANDLER
#endif
        @
        @ set desired IRQ state, then call main handler
        @
        msr       cpsr_c, r9
        mov       r2, sp
        bl        do_DataAbort

        @ IRQs off again before pulling preserved data off the stack
        @
        disable_irq

        @
        @ restore the registers and restart the instruction
        @
        ldmia     sp, {r0-r12}
        ldr       lr, [sp, #S_LR]
        add       sp, sp, #S_PC
        rfeia     sp!                   @ restore pc, cpsr
```

Sample – Exception Handler in ARM

```
        .macro  vector_stub, name, mode, correction=0
vector_\name:
        .if \correction
        sub     lr, lr, #\correction
        .endif

        @ Save r0, lr_<exception> (parent PC) and spsr_<exception>  (parent CPSR)

        stmia   sp, {r0, lr}            @ save r0, lr
        mrs     lr, spsr
        str     lr, [sp, #8]            @ save spsr

        @ Prepare for SVC32 mode.  IRQs remain disabled.

        mrs     r0, cpsr
        eor     r0, r0, #(\mode ^ SVC_MODE)
        msr     spsr_cxsf, r0

        @ the branch table must immediately follow this code

        and     lr, lr, #0x0f
        mov     r0, sp
        ldr     lr, [pc, lr, lsl #2]
        movs    pc, lr                  @ branch to handler in SVC mode
        .endm
[...]
        .macro  svc_entry
        sub     sp, sp, #S_FRAME_SIZE
        stmib   sp, {r1 - r12}

        ldmia   r0, {r1 - r3}
        add     r5, sp, #S_SP           @ here for interlock avoidance
        mov     r4, #-1                 @  ""  ""    ""    ""
        add     r0, sp, #S_FRAME_SIZE   @  ""  ""    ""    ""
        str     r1, [sp]                @ save the "real" r0 copied
                                        @ from the exception stack

        mov     r1, lr

        @ We are now ready to fill in the remaining blanks on the stack:
        @       r0 - sp_svc
        @       r1 - lr_svc
        @       r2 - lr_<exception>, already fixed up for correct return/restart
        @       r3 - spsr_<exception>
        @       r4 - orig_r0 (see pt_regs definition in ptrace.h)
        @
        stmia   r5, {r0 - r4}
        .endm

__dabt_svc:
        svc_entry

        @ get ready to re-enable interrupts if appropriate

        mrs     r9, cpsr
        tst     r3, #PSR_I_BIT
        biceq   r9, r9, #PSR_I_BIT

        @ Call the processor-specific abort handler:
        @       r2 - aborted context pc
        @       r3 - aborted context cpsr
        @
        @ The abort handler must return the aborted address in r0, and
        @ the fault status register in r1.  r9 must be preserved.

#ifdef MULTI_ABORT
        ldr     r4, .LCprocfns
        mov     lr, pc
        ldr     pc, [r4]
#else
        bl      CPU_ABORT_HANDLER
#endif

        @ set desired IRQ state, then call main handler

        msr     cpsr_c, r9
        mov     r2, sp
        bl      do_DataAbort

        @ IRQs off again before pulling preserved data off the stack

        disable_irq

        @ restore SPSR and restart the instruction

        ldr     r0, [sp, #S_PSR]
        msr     spsr_cxsf, r0
        ldmia   sp, {r0 - pc}^                  @ load r0 - pc, cpsr
```

Summary

- Thumb-2 core technology improves both ARM and Thumb ISAs to increase system performance and reduce cost.
- Thumb-2 core technology extends the Thumb ISA to provide a blended instruction set.
  - Average 20%better code density than ARM for Linux kernel and libraries using GCC
- With Thumb-2 developers don't have to manually balance between ARM and Thumb code
- Contribute kernel changes to mainline in 2007/2008
  - Thumb-2 support has been available with GNU compilation tools since 2006
- Higher code density can be achieved using optimized tool chains such as ARM RealView compilation tools