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

Connected Community
Development Tools
Software IP

Processors
System Level IP:
Data Engines
Fabric
3D Graphics
Physical IP
**Linux and ARM Processor Roadmap**

- **ARM11- MPCore 4-way**
  - Scalable.
  - IEM
  - AXI
- **ARM Cortex A8**
  - Super-scalar
  - Thumb-2
  - Neon
- **ARM Cortex M3**
- **ARM1136**
- **ARM1026 (area optimized)**
- **ARM1176**
  - TrustZone
  - IEM
  - AXI
- **Cortex-R4**
  - Scalable.
  - IEM
  - AXI
- **ARM1156**
  - Thumb-2
  - Fault-Tolerance
  - AXI
- **uClinux**

---

**Embedded Linux Conference - 2007**

**THE ARCHITECTURE FOR THE DIGITAL WORLD®**
# Processors Families

## Applications Processor Market

<table>
<thead>
<tr>
<th>Processor</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM11 MPCore</td>
<td>2000+ MIPS Multi-proc</td>
</tr>
<tr>
<td>ARM1176JZ(F)-S</td>
<td>600+ MIPS Uni-Proc</td>
</tr>
<tr>
<td>ARM1136J(F)-S</td>
<td>600+ MIPS Uni-Proc</td>
</tr>
<tr>
<td>ARM926EJ-S</td>
<td>250+ MIPS Uni-Proc</td>
</tr>
<tr>
<td>Cortex-A8</td>
<td>2000+ MIPS Uni-Proc</td>
</tr>
</tbody>
</table>

## Real-Time Embedded Market

<table>
<thead>
<tr>
<th>Processor</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM1156T2(F)-S</td>
<td>600+ MIPS Uni-Proc</td>
</tr>
<tr>
<td>ARM946E-S</td>
<td>150+ MIPS Uni-Proc</td>
</tr>
<tr>
<td>ARM968E-S</td>
<td>100+ MIPS Uni-Proc</td>
</tr>
<tr>
<td>ARM7TDMI</td>
<td>Cortex R4(F) 600+ MIPS Uni-Proc</td>
</tr>
</tbody>
</table>

## Microcontroller Market

<table>
<thead>
<tr>
<th>Processor</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI</td>
<td>Cortex-M3</td>
</tr>
</tbody>
</table>

---

*Image credit: ARM*
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 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

- 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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>not 111</td>
<td>xx</td>
<td>16 bits</td>
<td>Current 16-bit Thumb instruction</td>
</tr>
<tr>
<td>111</td>
<td>00</td>
<td>16 bits</td>
<td>Current 16-bit Thumb B unconditional</td>
</tr>
<tr>
<td>111</td>
<td>not 00</td>
<td>32 bits</td>
<td>Thumb-2 32-bit</td>
</tr>
</tbody>
</table>

- 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
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

<table>
<thead>
<tr>
<th>ARM* or Thumb-2</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFI R0, R1, #bitpos, #fieldwidth</td>
<td>AND R2, R1, #bitmask</td>
</tr>
<tr>
<td></td>
<td>BIC R0, R0, #bitmask &lt;&lt; bitpos</td>
</tr>
<tr>
<td></td>
<td>ORR R0, R0, R2, LSL #bitpos</td>
</tr>
</tbody>
</table>
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”

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

<table>
<thead>
<tr>
<th>ARM</th>
<th>Thumb-2</th>
<th>Thumb</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP r0, #0</td>
<td>CBZ r0, ln</td>
<td>CMP r0, #0</td>
</tr>
<tr>
<td>BEQ ln</td>
<td></td>
<td>BEQ ln</td>
</tr>
<tr>
<td>8 Bytes, 1 or 2 cycles</td>
<td>2 Bytes, 1 cycle</td>
<td>4 Bytes, 1 or 2 cycles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARM</th>
<th>Thumb-2</th>
<th>Thumb</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDREQ r0, [r1]</td>
<td>ITETE EQ</td>
<td>BNE I1</td>
</tr>
<tr>
<td>LDRNE r0, [r2]</td>
<td>LDREQ r0, [r1]</td>
<td>LDR r0,[r1]</td>
</tr>
<tr>
<td>ADDEQ r0, r3, r0</td>
<td>LDRNE r0, [r2]</td>
<td>ADD r0, r3, r0</td>
</tr>
<tr>
<td>ADDNE r0, r4, r0</td>
<td>ADDEQ r0, r3, r0</td>
<td>B I2</td>
</tr>
<tr>
<td>ADDNE r0, r4, r0</td>
<td>ADDNE r0, r4, r0</td>
<td>I1 LDR r0,[r1]</td>
</tr>
<tr>
<td></td>
<td>ADD r0, r4, r0</td>
<td>ADD r0, r4, r0</td>
</tr>
<tr>
<td></td>
<td>I2 ........</td>
<td></td>
</tr>
<tr>
<td>16 Bytes, 4 cycles</td>
<td>10 Bytes, 4 or 5 cycles</td>
<td>12 Bytes, 4 to 20 cycles</td>
</tr>
</tbody>
</table>
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
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

<table>
<thead>
<tr>
<th></th>
<th>ARM Mode</th>
<th>Thumb-2 mode</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>libc-2.3.6.so</td>
<td>1123552</td>
<td>824544</td>
<td>73%</td>
</tr>
<tr>
<td>libm-2.3.6.so</td>
<td>669496</td>
<td>542520</td>
<td>81%</td>
</tr>
<tr>
<td>2.6.19 kernel</td>
<td>1019832</td>
<td>724888</td>
<td>71%</td>
</tr>
<tr>
<td>MPlayer (dynamic)</td>
<td>5793064</td>
<td>5619000</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>6707792 (static)</td>
<td>5176036 (static)</td>
<td>77%</td>
</tr>
</tbody>
</table>
Sample - Exception Handler in Thumb-2

```assembly
.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
  rsadb    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\_cxs, 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
  ...

  @ 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
  ...

  .macro svc_entry
  stmia    sp, {r0 - r12}
  .endm

__dabt_svc:
  svc\_entry
  ...
```

```assembly
@
@ get ready to re-enable interrupts if appropriate
@
mrs      r9, cpsr
  
tst      r3, #PSR\_I\_BIT
  
it       eq
  
bl        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, \_Lprocfn
  mov      lr, pc
  ldr      pc, [r4]
#else
  bl       CPU\_ABORT\_HANDLER
@end if

@  @ set desired IRQ state, then call main handler
@
mrs      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

```assembly
...}
...

__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, .LProcfn

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
```

@ 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

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)
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