Introduction to A64 Instruction Set

Rodolph Perfetta
Software & System Group, ARM®
About Me

- Software Engineer
- GoogleV8 JavaScript Engine
  - A64 initial port
- VIXL
  - A64 runtime assembler/disassembler/simulator
- Java VM
  - When memory unit was KB
AArch64

- New ISA: A64
  - Similar functionality to ARM®/Thumb2®
- 64-bit registers
- 64-bit pointers (48-bit payload)
- 32-bit instructions (fixed length)
- Floating point and SIMD mandatory
  - IEEE FP math in SIMD
- Little Endian (Big Endian is possible)
- Weakly ordered memory (like ARMv7)
  - Don’t forget barriers
AArch32

- Crypto extension
- New FP instructions
- Deprecated Instructions
  - SETEND
  - IT, partially
- Obsolete Instructions
  - SWP, SWPB
  - VFP short vectors
  - CP15 barriers
Registers

- AArch64 provides 32 registers, of which 31 are general purpose
  - Each register has a 32-bit ($w_0$–$w_{30}$) and 64-bit ($x_0$–$x_{30}$) form
  - Writing to a W register clears the top 32 bits of the corresponding X register
  - $x_{31}$ is either a “zero” register ($xzr$, $wzr$), or the stack pointer ($sp$, $wsp$) depending on the instruction

![Diagram of registers](image)
Registers

- Most instructions can be 32 or 64-bit
  - `add w0, w1, w2`
  - `add x0, x1, x2`

- Be careful
  - `add w1, w1, #0` // this is not a nop, clear top 32-bit of x1
  - `str x31, [...]` // store zero
  - `ldr x31, [...]` // ignore the result of the load
Floating-point registers

- Separate register file for floating point, SIMD and crypto operations - Vn
  - 32 registers, each 128-bits
    - Can also be accessed in 32-bit (Sn) or 64-bit (Dn) forms

- No more overlapping!
Special registers

- The PC is not a general purpose register
  - `adr` instruction can be used to get the address of a PC relative offset
    
    `adr x0 label`

- `x31` is the stack pointer and must always be 128-bit aligned
  - Hardware checking of SP alignment is enforced

- There are also some ABI defined registers:
  - `x30` is LR
    - Updated by branch with link instructions (e.g. `BL`)
  - `x29` is FP
extern int foo(int, int);

int main(void)
{
    ...
    a = foo(b, c);
    ...
}
Instructions

- Only a few instructions can set flags
  - `adds`, `ands`, `subs` and aliases.

- Arithmetic operations: `add`, `sub`, `adc`, `neg`, `mul`, `sdiv`...
- Logical operations: `and`, `orr`, `eor`, `bic`...
- Shift and rotate: `lsl`, `lsr`, `asr`, `ror`...
- Sign/Zero extension: `sxtb`, `sxtw`, `uxtb`...
- Bit manipulation: `bfm`, `sbfm`, `bfi`...
- Branch: `b`, `bl`, `br`, `blr`, `cbz`, `tbz`...
- Load/Store: `ldr`, `str`...
- Conditional
  - Not like A32 conditionals
# Load and Store

- No Load/Store Multiple instructions anymore
  - We have load pair and store pair instead (ldp/stp)

<table>
<thead>
<tr>
<th>A32</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmdb sp!, {r0, r1, r2, r3, r4, r5, r6</td>
</tr>
<tr>
<td>r7, r8, r9, r10, r11, r12}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A64</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub sp, sp, #frame_size</td>
</tr>
<tr>
<td>stp x0, x1, [sp, #0]</td>
</tr>
<tr>
<td>stp x2, x3, [sp, #16]</td>
</tr>
<tr>
<td>stp x4, x5, [sp, #32]</td>
</tr>
<tr>
<td>stp x6, x7, [sp, #48]</td>
</tr>
<tr>
<td>stp x8, x9, [sp, #64]</td>
</tr>
<tr>
<td>stp x10, x11, [sp, #80]</td>
</tr>
<tr>
<td>stp x12, x13, [sp, #96]</td>
</tr>
<tr>
<td>stp x14, x15, [sp, #112]</td>
</tr>
<tr>
<td>stp x18, x19, [sp, #128]</td>
</tr>
<tr>
<td>stp x20, x21, [sp, #144]</td>
</tr>
<tr>
<td>stp x22, x23, [sp, #160]</td>
</tr>
<tr>
<td>stp x24, x25, [sp, #176]</td>
</tr>
<tr>
<td>stp x26, x27, [sp, #192]</td>
</tr>
<tr>
<td>stp x28, x29, [sp, #208]</td>
</tr>
</tbody>
</table>
Conditional Instructions

- Conditional select
  - Set a register to one of its inputs depending of the condition
    \[
    \text{csel } x0, x1, x2, \text{ eq}
    \]

<table>
<thead>
<tr>
<th></th>
<th>A32</th>
<th>A64</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp</td>
<td>r0, #value</td>
<td>cmp x0, #value</td>
</tr>
<tr>
<td>mov</td>
<td>r0, r1, eq</td>
<td>csel x0, x1, x2, eq</td>
</tr>
<tr>
<td>mov</td>
<td>r0, r2, ne</td>
<td></td>
</tr>
</tbody>
</table>

- Additional uses:
  \[
  \text{csel } x10, xzr, x10, \text{ cond} \quad \text{// Conditional clear}
  \]

- Other instructions: \text{CINC, CSET, CNEG, CCMP} ...
Thanks

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.