Introduction to A64 Instruction Set

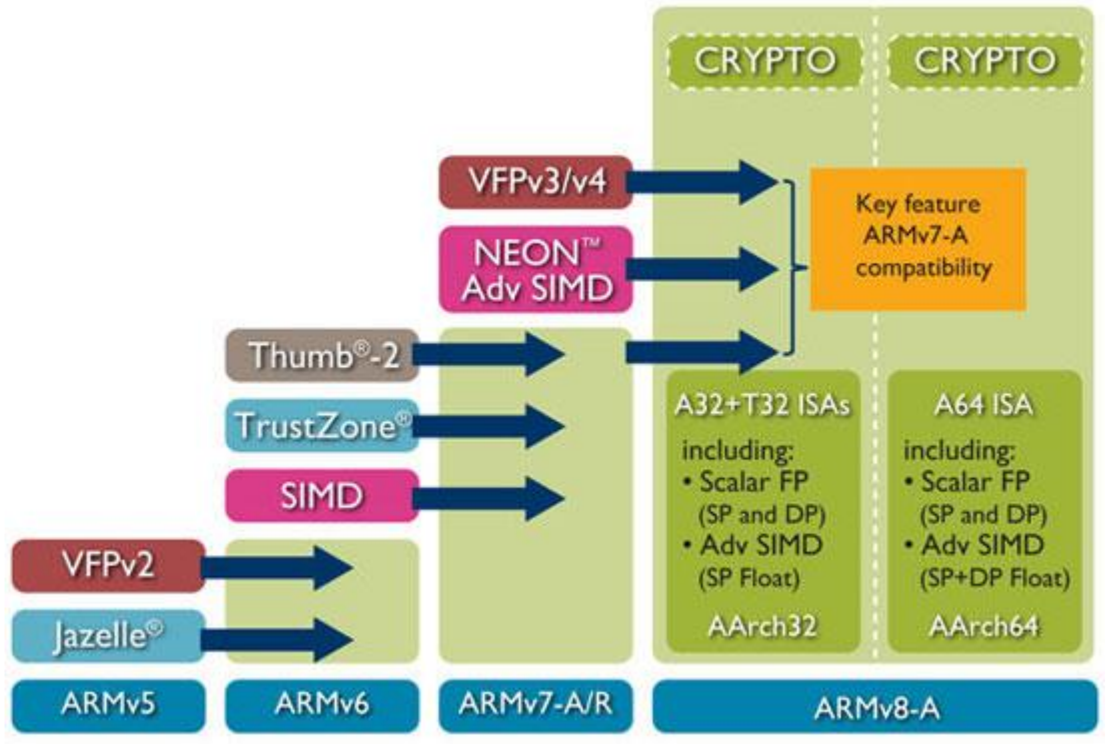
Rodolph Perfetta

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

ARMv8


CRYPTO
CRYPTO
VFPv3/v4
NEON™ Adv SIMD
Key feature ARMv7-A compatibility
Thumb®-2
TrustZone®
SIMD
VFPv2
Jazelle®
A32+T32 ISAs
including:
• Scalar FP (SP and DP)
• Adv SIMD (SP Float)
AArch32
A64 ISA
including:
• Scalar FP (SP and DP)
• Adv SIMD (SP+DP Float)
AArch64
ARMv5
ARMv6
ARMv7-A/R
ARMv8-A

AArch64

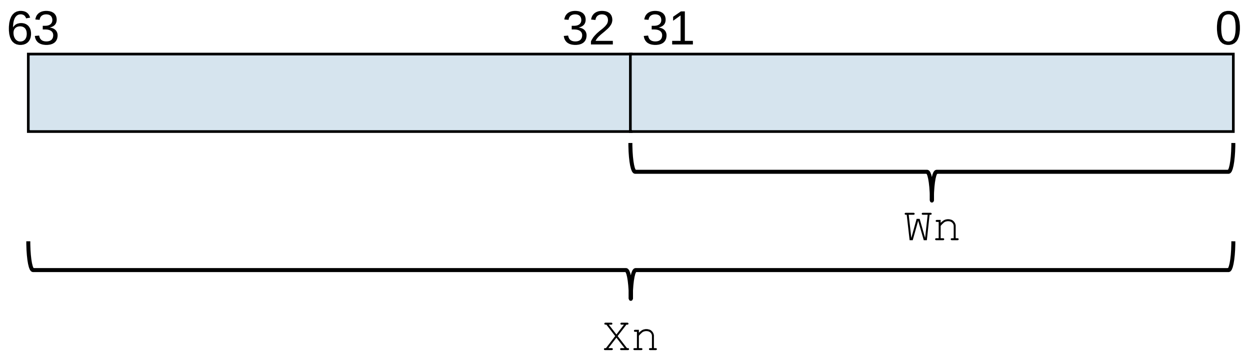
- New ISA: A64
  - Similar functionality to ARM®/Thumb2®
- 64-bit registers
- 64-bit pointers (48-bit payload)
- 32-bit instructions (fixed length)
- Floating point and SIMD mandatory
  - IEEE FP math in SIMD
- Little Endian (Big Endian is possible)
- Weakly ordered memory (like ARMv7)
  - Don’t forget 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 (`w0-w30`) and 64-bit (`x0-x30`) form
  - Writing to a W register **clears the top 32 bits** of the corresponding X register
  - `x31` is either a "zero" register (`xzr, wzr`), or the stack pointer (`sp, wsp`) depending on the instruction

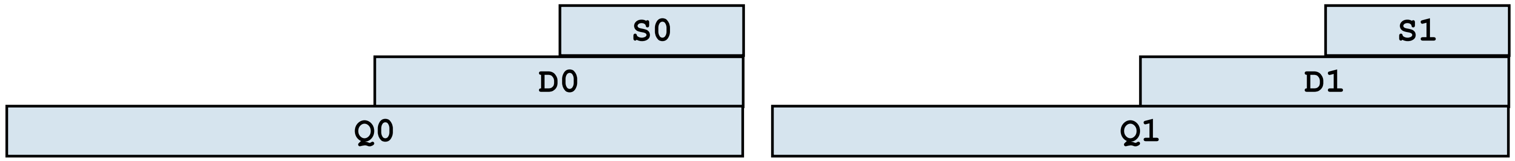


Registers

- 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
    ```

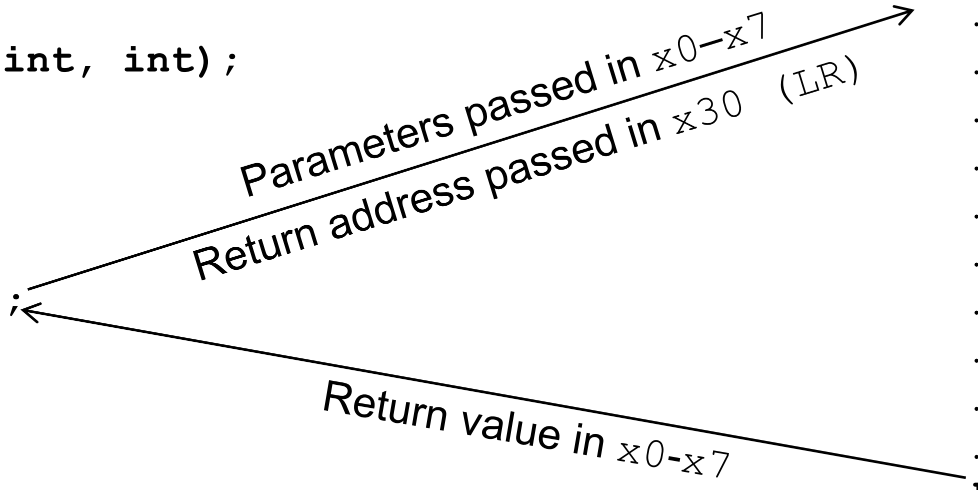
- x31 is the stack pointer and must always be 128-bit aligned
  - Hardware checking of SP alignment is enforced

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

Procedure call standard

```
extern int foo(int, int);

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



```
   EXPORT foo
foo PROC
    …
    …
    …
    …
    …
    …
    …
    …
    …
    …
    RET
    ENDP
```

Must preserve
X19 - X29
D8 - D15

Can corrupt:
X0 - X18
D0 - D7
D16 - D31

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`)

A32	A64

`stmdb sp!, {r0, r1, r2, r3, r4, r5, r6`
`r7, r8, r9, r10, r11, r12}`
	`sub sp, sp, #frame_size`
	`stp x0, x1, [sp, #0]`
	`stp x2, x3, [sp, #16]`
	`stp x4, x5, [sp, #32]`
	`stp x6, x7, [sp, #48]`
	`stp x8, x9, [sp, #64]`
	`stp x10, x11, [sp, #80]`
	`stp x12, x13, [sp, #96]`
	`stp x14, x15, [sp, #112]`
	`stp x18, x19, [sp, #128]`
	`stp x20, x21, [sp, #144]`
	`stp x22, x23, [sp, #160]`
	`stp x24, x25, [sp, #176]`
	`stp x26, x27, [sp, #192]`
	`stp x28, x29, [sp, #208]`

Conditional Instructions

- Conditional select
  - Set a register to one of its inputs depending of the condition

```
csel x0, x1, x2, eq
```

A32	A64

`cmp   r0, #value`
`mov   r0, r1, eq`
`mov   r0, r2, ne`
	`cmp   x0, #value`
	`csel x0, x1, x2, eq`

  - Additional uses:

```
csel x10, xzr, x10, cond    // Conditional clear
```

- Other instructions: `CINC, CSET, CNEG, CCMP` ...

Thanks