原创 无色云 于 2020-07-11 12:23:36 发布 3315 收藏 17 版权
分类专栏: MIPS/ARM体系结构/汇编

有了之前mips系统架构的基础，再了解arm64就相对轻松多了。所谓体系架构，核心就是寄存器、指令集和abi（即寄存器和指令集的使用规范）。下面就分这几方面展开做的学习笔记。

目录

一、armv8概览

ARM架构版本号从1-8。ARMv8架构支持一下两种执行状态：

aarch64:

1、提供31个64位通用寄存器（其中X30被用来做链接寄存器LR（函数返回地址））。和一个64位程序计数器PC、栈帧SPs、异常链接寄存器ELRs。
3、提供32个128-bit寄存器用于SIMD vector and scalar floating-point support.
4、仅支持指令集 A64.
5、4个异常等级 EL0 - EL3
6、Defines a number of Process state (PSTATE) elements that hold PE state. The A64 instruction set includes instructions that operate directly on various PSTATE elements.
7、Names each System register using a suffix that indicate the register can be accessed.

1、提供13个32-bit通用寄存器,和一个32-bit PC, SP, 和链接寄存器 (LR)。LR 还兼有 ELR 功能。

2、提供32个64-bit寄存器用于SIMD vector and scalar floating-point support.

3、支持两种指令集 A32 （32bit编码的定长指令集）和 T32（使用16bit和32bit编码的变长指令集）。

二、ARM64寄存器

ARM64寄存器分类：通用寄存器、 浮点寄存器、状态寄存器、协处理器寄存器。本文就通用寄存器的使用来介绍。其他寄存器根据后期学习情况逐步补充。

1、ARM64通用寄存器

通用寄存器就是指用户态可以使用的寄存器。它和我们普通程序员关系最为密切。下表列举了ARM64的通用寄存器。

寄存器	描述
x0-x7	用于子程序调用时的参数传递，X0还用于返回值传递
x8-x15	临时寄存器，也叫可变寄存器，子程序使用时不需要保存。
x16-x17	子程序内部调用寄存器（IPx），使用时不需要保存，尽量不要使用
x18	平台寄存器，它的使用与平台相关，尽量不要使用。
x19-x28	临时寄存器，子程序使用时必须保存。
x29/fp	用于连接栈帧，使用时必须保存。
x30/LR	存放的是函数的返回“地址”。当ret指令执行时刻,会寻找x30寄存器保存的“地址值”!
SP	用于指向每个函数的栈顶。32位栈帧使用WSP。
XZR	零寄存器。32位零寄存器为WZR

说明： x0 - x30 是31个通用整形寄存器。每个寄存器可以存取一个64位大小的数。 当使用 x0 - x30 访问时，它就是一个64位的数。当使用 w0 - w30 访问时，访问的是这些寄存器的低32位，如图：

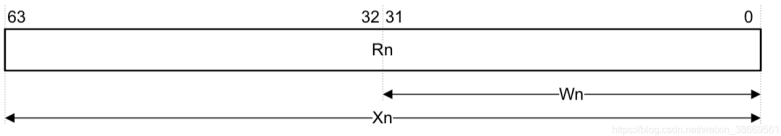
Registers in AArch64 state

In the AArch64 application level view, an ARM processing element has:

R0-R30 31 general-purpose registers, R0 to R30. Each register can be accessed as:

- A 64-bit general-purpose register named X0 to X30.
- A 32-bit general-purpose register named W0 to W30.

See the register name mapping in Figure B1-1.



2、状态寄存器

ARM体系架构中有一个当前程序状态寄存器CPSR（R16？？）和5个备份状态寄存器SPSRs。先关注CPSR，CPSR可以在任何工作模式下访问，用来保存ALU中的当前信息、状态等。基本格式如下：

31 26 8

如上图所示，CPSR分为3部分：条件标志位NZCVQ、控制位IFTM4M3M2M1M0、保留位。

2.1 条件标志位

上图蓝色部分N，Z，C，V，Q为条件码标志位。它们内容可以根据算术或逻辑运算的结果所改变，并用来决定某条指令是否被执行。具体例子见 1.4 条件后缀 . 条件码标志的具体含义如下表：

标志位	含义
N	正负标志。N=1标识运算结果为负数
Z	零标志。Z=1标识运算结果为0
C	进位标志。加法运算结果有进位则C=1;
V	溢出标志。运算结果有溢出则V=1
Q	？？

2.2 控制位

~~略，后期需要时补上~~

2.3 保留位

~~略，后期需要时补上~~

3、32 个SIMD&FP 寄存器 V0 -V31

每个寄存器根据实际使用长度，命名会有所不同。具体表示如下：

- A 128-bit register named Q0 to Q31.
- A 64-bit register named D0 to D31.
- A 32-bit register named S0 to S31.
- A 16-bit register named H0 to H31.
- An 8-bit register named B0 to B31.
- A 128-bit vector of elements.
- A 64-bit vector of elements.
- 如下图所示：

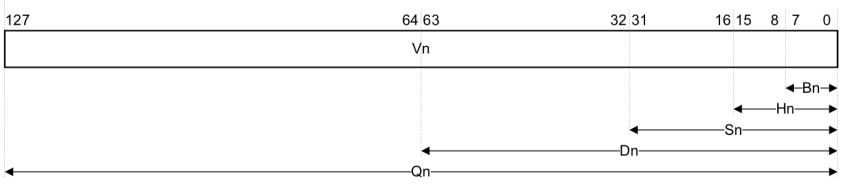


Figure B1-2 SIMD and floating-point register naming

4、2 个SIMD&FP 控制寄存器 FPCR 、FPSR

- Two SIMD and floating-point control and status registers, FPCR and FPSR.

三、Aarch64指令集

为ARM特有，MIPS没有此类指令。下面分别介绍。

学习指令之前要了解一下指令后缀。

1、指令后缀

ARM中的指令可以带后缀，从而丰富该指令的功能，常用的后缀有：

1.1 位数

B(byte)：功能不变，操作长度变为8位(依赖CPU位数，以下相同)

H(Halfword)：功能不变，操作长度变为16位

例如： ldr指令族：ldrb,ldrh,ldrsb ldrsh，从内存中加载指定长度的数据

1.2 S(S标识)：影响CPSR里的NZCV标识位

指令中使用"S"后缀时、指令执行后程序状态寄存器的条件标志位将被刷新。相当于有符号运算。例如：

SUB X1，X0，X3 ;X1=X0-X3 ，CPSR值不变

SUBS X1，X0，X3 ;X1=X0-X3 , 如果计算结果为负数，CPSR寄存器的N被置位

1.3 "！"后缀

指令中地址表达式含有"！"后缀时，指令执行后，基址寄存器中的地址值将会发生变化。变化的结果是 (base+offset) 。例如：

LDR X3，[X0,#4] //X3=X0+4

LDR X3，[X0,#4]! //X3=X0+4; X0+=4；

注意： "！"不能用于寄存器PC后面

1.4 条件后缀

ARM架构中，允许在指令后面添加条件后缀来完成指令条件执行的目的。指令条件执行就是说，指令根据CPSR中条件码的状态和指令的条件域有条件的执行。当指令的执行条件满足时，指令被执行，否则指令将被忽略。例如比较下面两条指令：

ADD X4，X2，#1 ；无条件执行 X4=X2+1

ADDEQ X4，X2，#1 ;添加有条件执行后缀EQ，当CPSR中的Z标志置位（之前某条CMP结果相等）时，该指令才执行。

注意：如果条件后缀和"S"标识同时出现，则S在条件后缀的后面，例如：

ADDEQS X4，X2，#1 ;即为有条件执行X4=X2+1，结果更新条件标志位

ARM中支持的条件后缀（条件码）共16种（系统保留1种，剩下15种可用），具体标志和功能如下表：

指令码	含义
EQ	Z置位 ;结果相等才执行
NE	Z清零，结果不相等才执行
CS	C置位，结果无符号>= 才执行
CC	C清零，结果无符号< 才执行
MI	N置位，结果为负数才执行
PL	N清零，结果为正数或0才执行
VS	V置位，结果溢出才执行

HI	C置位Z清零，结果为无符号数大于才执行
LS	C清零Z置位，结果为无符号数小于或等于才执行
GE	N等于V，结果为有符号数大于或等于才执行
LT	N不等于V，结果为有符号数小于才执行
GT	Z清零且N等于V，结果为有符号大于才执行
LE	Z置位或N不等于V，结果为有符号数小于或等于
AL	无条件执行。省略。

例如：

```
cmp    w0, #0    ;
cset   x0, lt        ;如果(w0 < 0)，X0置1，否则X0置0
```

说明：指令编程风格

ARM官方风格：官方风格指令一般使用大写，例如：LDR R0,[R1]，Windows中常使用这种风格。

GUN Linux风格：指令一般使用小写字母，例如：ldr r0,[r1]，Linux环境中常用这种风格.

本文下面的介绍多以官方风格来编写指令。

2、数据处理（运算）指令：

2.1 数据传送指令

指令	功能	格式
MOV	赋值	MOV Wd|WSP, Wn|WSP ; 32-bit MOV Xd|SP, Xn|SP ; 64-bit
MVN	取反赋值	MVN Wd, Wm{, shift #amount} ; 32-bit MVN Xd, Xm{, shift #amount} ; 64-bit

例如：

MOV X1，X0 ;将寄存器X0赋值给X1

MVN X1，X2 ;将寄存器X2取反后传送给寄存器X1

2.2 移位指令

指令	功能	格式
LSL	逻辑左移	LSL Wd, Wn, Wm ; 32-bit LSL Xd, Xn, Xm ; 64-bit
LSR	逻辑右移	LSR Wd, Wn, Wm ; 32-bit LSR Xd, Xn, Xm ; 64-bit
ASR	算术右移	ASR Wd, Wn, Wm ; 32-bit ASR Xd, Xn, Xm ; 64-bit
ROR	循环右移	ROR Wd, Wn, Wm ; 32-bit ROR Xd, Xn, Xm ; 64-bit

例如：

```
LSL    w1, w1, #1        ;将寄存器W1逻辑左移1位后，赋值给W1
MOV X0，X1，LSL#1   ;将寄存器X1左移1位后赋值给X0。
```

2.3 算术运算指令

指令	功能	格式
add	加法运算	ADD Wd|WSP, Wn|WSP, #imm{, shift} ; 32-bit ADD Xd|SP, Xn|SP, #imm{, shift} ; 64-bit
sub	减法运算	SUB Wd|WSP, Wn|WSP, Wm{, extend {#amount}} ; 32-bit SUB Xd|SP, Xn|SP, Rm{, extend {#amount}} ; 64-bit
rsb	反减运算	
adc	带进位的加法运算	ADC Wd, Wn, Wm ; 32-bit ADC Xd, Xn, Xm ; 64-bit
sbc	带进位的减法运算	SBC Wd, Wn, Wm ; 32-bit SBC Xd, Xn, Xm ; 64-bit

例如：

ADC X1，X3，X5 ;X1=X3+X5+C（CPSR中的C位）

2.4 逻辑运算指令

指令	功能	格式
and	与操作	AND Wd|WSP, Wn, #imm ; 32-bit AND Xd|SP, Xn, #imm ; 64-bit 操作：Rd = Rn & imm ,where R is either W or X .
orr	或操作	ORR Wd|WSP, Wn, #imm ; 32-bit ORR Xd|SP, Xn, #imm ; 64-bit 操作：Rd = Rn | imm ,where R is either W or X .
eor	异或操作	EOR Wd|WSP, Wn, #imm ; 32-bit EOR Xd|SP, Xn, #imm ; 64-bit 操作 Rd = Rn ^ imm ,where R is either W or X
bic	位清除操作	BIC Wd, Wn, Wm{, shift #amount} ; 32-bit BIC Xd, Xn, Xm{, shift #amount} ; 64-bit 操作：Rd = Rn AND NOT shift(Rm, amount) ,where R is either W or X .

例如：

BIC X0，X0 #9 ;这里十进制数9对应的二进制数为101 ，故实现将寄存器X0的第0位和第3位清零。

2.5 比较运算指令

指令	功能	格式
CMP	比较大小	CMP Wn|WSP, Wm{, extend {#amount}} ; 32-bit CMP Xn|SP, Rm{, extend {#amount}} ; 64-bit
CMN	反值比较	CMN Wn|WSP, Wm{, extend {#amount}} ; 32-bit CMN Xn|SP, Rm{, extend {#amount}} ; 64-bit
TST	按位与运算	TST Wn, #imm ; 32-bit TST Xn, #imm ; 64-bit 即 Rn AND imm where R is either W or X .

例如：

CMP X1，#10 ;根据（R1-10）的结果更改CPSR的标志位

CMN W0，W1 ；根据（W0+W1）的结果更改CPSR的标志

2.6 乘除运算指令

指令	功能	格式
mul	乘法运算。	MUL Wd, Wn, Wm ; 32-bit MUL Xd, Xn, Xm ; 64-bit
UMULL	64位无符号乘法	UMULL Xd, Wn, Wm => Xd = Wn * Wm
smull	有符号乘法	SMULL Xd, Wn, Wm 等同于 SMADDL Xd, Wn, Wm, XZR .
SMADDL	有符号乘加	SMADDL Xd, Wn, Wm, Xa => Xd = Xa + Wn * Wm
UDIV	无符号除法	UDIV Wd, Wn, Wm ; 32-bit UDIV Xd, Xn, Xm ; 64-bit
SDIV	有符号除法	SDIV Wd, Wn, Wm ; 32-bit SDIV Xd, Xn, Xm ; 64-bit

说明：ARM64中乘法，根据源操作数也分为32位64位乘法。由于64位乘法结果太大，需要放在2个64位寄存器Rdlo和寄存器Rdhi。寄存器Rdlo用于存放结果的低64位，寄存器Rdhi用于存放结果的高64位。类似于MIPS中的乘法寄存器HI和LO。

注意1：目的寄存器rd和操作数寄存器rm必须是不同的寄存器

3、数据加载/存储指令

ARM64的数据加载/存储，类似于MIPS体系架构的load/store。

指令	功能	格式
LDR	32/64位加载寄存器	LDR Wt, [Xn|SP, (Wm|Xm){, extend {amount}}] ; 32-bit LDR Xt, [Xn|SP, (Wm|Xm){, extend {amount}}] ; 64-bit

LDRB	加载一个字节	LDRB Wt, [Xn|SP], #simm ; Post-index general registers
		LDRB Wt, [Xn|SP, #simm]! ; Pre-index general registers
		LDRB Wt, [Xn|SP{, #pimm}] ; Unsigned offset general registers

LDRH	加载半字	LDRH Wt, [Xn|SP], #simm ; Post-index general registers
		LDRH Wt, [Xn|SP, #simm]! ; Pre-index general registers
		LDRH Wt, [Xn|SP{, #pimm}] ; Unsigned offset general registers

LDRSB	加载有符号字节	LDRSB Wt, [Xn|SP], #simm ; 32-bit, Post-index LDRSB Xt, [Xn|SP], #simm ; 64-bit, Post-index LDRSB Wt, [Xn|SP, #simm]! ; 32-bit, Pre-index LDRSB Xt, [Xn|SP, #simm]! ; 64-bit, Pre-index LDRSB Wt, [Xn|SP{, #pimm}] ; 32-bit LDRSB Xt, [Xn|SP{, #pimm}] ; 64-bit
STR	32/64位存储	STR Wt, [Xn|SP, (Wm|Xm){, extend {amount}}] ; 32-bit STR Xt, [Xn|SP, (Wm|Xm){, extend {amount}}] ; 64-bit
STRH	半字存储	STRH Wt, [Xn|SP, (Wm|Xm){, extend {amount}}]
SWP	交换指令	

例如：

LDR X3，[X2,#0x8] ；load memory(X2+0x8) to X3

STR X6,[X5,X3] ；store X6 to memory(X5+X3)

SWP X0，X1，[X2] ;将memory(X2)数据加载到X0，同时将X1中的数据存储到memory(X2)

注意：没有找到教材中说的多寄存器加载/存储指令 LDM/ST

类似于MIPS架构的B/J指令。用于分支跳转和函数调用。

指令	功能	格式
B	无条件跳转，不返回 in the range ±128MB	B label
BL	跳转前把返回地址放入x30(LR)中 range ±128MB	setting the register X30 to PC+4 BL label
BR	跳转到寄存器	BR Xn
BLR	寄存器跳转。跳转前把返回地址放入x30(LR)中 长跳转range 2的64次方-1	BLR Xn
BX	跳转同时切换到ARM模式，用于异常处理的跳转	?
CBZ	比较Rt结果为零（Zero）则跳转	CBZ Rt, label ; 64-bit R为Xt或者Wt
BEQ	CPSR中Z置位则跳转	BEQ label ; UAL(A32/T32) B.EQ label ; A64

例如：

```
cmp    w0, #15    //将(w0-15)结果更新CPSR标志位
beq    label      //如果(w0==15) ,则跳转到lable
```

5、程序状态寄存器的访问指令

由于程序状态寄存器不属于通用寄存器，故不能直接读取，而是要通过下面两条指令专门读取。

指令	功能	格式
MRS	读状态寄存器	MRS Xt, (systemreg|Sop0_op1_Cn_Cm_op2)
MSR	写状态寄存器	MSR pstatefield, #imm MSR (systemreg|Sop0_op1_Cn_Cm_op2), Xt

例如：

MRS X0，SPSR ;读SPSR的内容到X0

MSR CPSR_c,X0 ;写寄存器X0值到CPSR，仅修改CPSR的控制位域c

6、协处理器指令

~~略，后期需要时补上~~

指令	功能	格式
CDP		
LDC		
STC		
MCR		
MRC		

7、软中断指令

FIXME： SVC was called SWI in earlier versions of the A32 disassemble to SVC ,

指令	功能	格式
SVC	软件中断	SVC{cond} #imm

swi(software interrupt)，在软件层模拟产生一个中断，这个中断会传送给CPU，常用于实现系统调用。类似于mips架构的syscall指令。

~~待确认，如何使用此指令，根据系统调用号ID来实现系统调用~~

8、其他指令

暂时把无法归类的指令称为其他指令吧。

指令	功能	格式
BRK	ARM64的断点指令，imm标识一个不大于16位的立即数	BRK #imm
BKPT	ARM32的断点指令，imm标识一个不大于16位的立即数	BKPT #imm
clz	统计一个数的二进制位前面有几个0	CLZ{cond} Rd, Rm

断点指令BRK/BKPT可以触发SIGTRAP中断， 类似于mips架构的break指令。

~~缺少示例~~

四、使用规范ABI

~~略，后期需要时补上~~