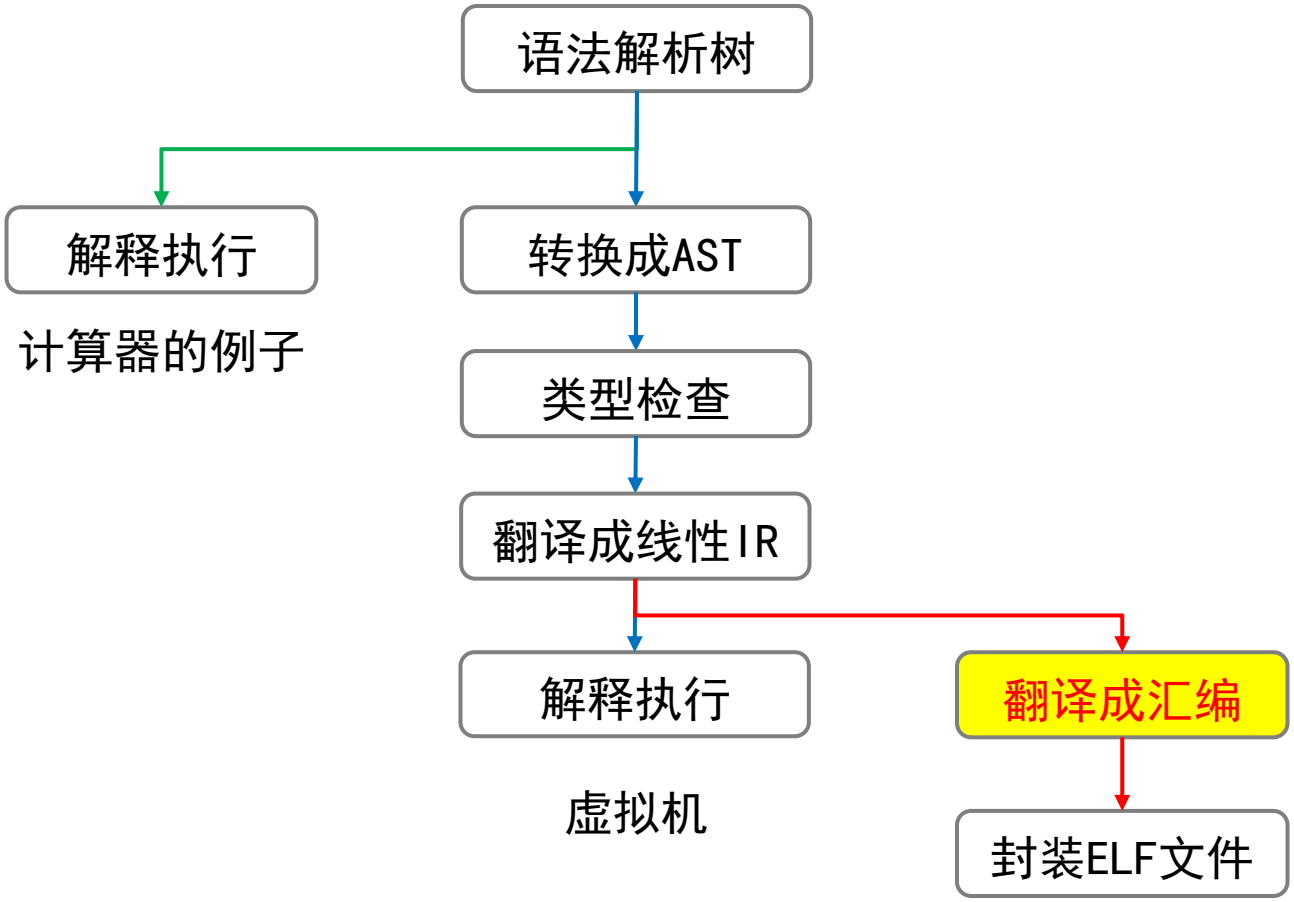
Lecture 12

汇编代码

徐 辉
xuh@fudan.edu.cn

编译地图


语法解析树
解释执行
计算器的例子
转换成AST
类型检查
翻译成线性IR
解释执行
虚拟机
翻译成汇编
封装ELF文件

大纲

- 一、基础arm指令
- 二、调用规约
- 三、更多指令和寄存器

一、基础arm指令

指令集架构：Instruction Set Architecture

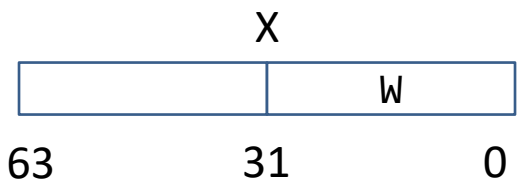
- 精简指令集（RISC）
  - 每条指令只做一件事（主要指内存访问和运算分离）
  - ARM、RISC-V
- 复杂指令集（CISC）
  - 一条指令可以包含多个底层操作（load-add-store）
  - X86（Intel IA-32）、X86-64架构（AMD64）
- 其它
  - VLIW：Very long instruction word (Intel IA-64)
  - EPIC：Explicitly parallel instruction computing

目标指令集：arm v8a

- 32位版本：ARM Cortex-A32
- 32/64位版本：ARM Cortex-A57/A72/A73等，代表芯片
  - Apple A8/A8x/A9/A9x/A10/A10x（iPhone, iPad）
  - Apple M1/M2（iPhone, iPad, MacBook）
  - Qualcomm Kryo: 骁龙（Snapdragon 820）
- aarch64 vs arm64 (Apple版本)

arm v8a寄存器：通用寄存器

- 通用寄存器：X0 - X30 (64-bit)




```
mov x1, #5
mov x2, #10
add x0, x1, x2
```

```
x1 = 5
x2 = 10
x0 = x1 + x2
```

```
mov w1, #5
mov w2, #10
add w0, w1, w2
```

```
w1 = 5
w2 = 10
w0 = w1 + w2
```

IR=>Assembly：数据存取

- 栈（顶）寄存器：SP（16字节对齐）

```
%x = alloca i32
store i32 1, %x
%x0 = load i32, i32* %x
```

```
sub sp, sp, #16
mov w0, #1
str w0, [sp]
ldr w0, [sp]
```

sp = sp - 16

```
#: llc -O0 -march=arm64 -filetype=asm foo.ll –o foo.s
#: as foo.s –o foo.o
```

MacOS (M1)

```
#: ld foo.o -lSystem -syslibroot `xcrun -sdk macosx --show-sdk-path` -e _start -arch arm64
```

或

```
#: gcc test.o -e _main -arch arm64
```

arm v8a指令：寻址模式

- 不支持直接寻址，间接寻址

Addressing Mode	Offset		
	Immediate	Register	Extended Register
Base register only (no offset)	[base{, #0}]	-	-
Base plus offset	[base{, #imm}]	[base, Xm{, LSL #imm}]	[base, Wm, (S|U)XT(X|W) {#imm}]
Pre-indexed	[base, #imm]!	-	-
Post-indexed	[base], #imm	[base], Xm[a]	-
Literal (PC-relative)	label	-	-

```
ldr x2, [x1]
ldr x2, [x1, #10]
ldr x2, [x1, x0]
ldr x2, [x1, #10]!
ldr x2, [x1], #10
ldr x2, [x1, x0, lsl #3]
```

```
x2 = [x1 + 10]
x2 = [x1 + x0]
x1 = x1 + 10, x2 = [x1]
x2 = [x1], x1 = x1 + 10
x2 = [x1 + x0 *3]
```

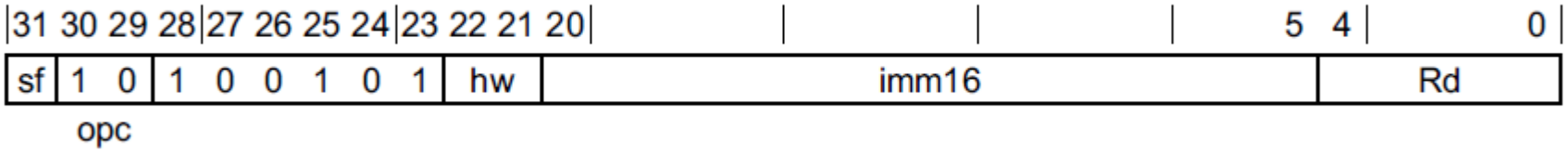
```
str w0, [x1]
str w0, [x1, #10]
str x2, [x1, x0]
```

arm v8a指令：MOV

- MOV：任意16位立即数，或左移若干位？

MOV <Xd|SP>, <Xn|SP>　　ADD <Wd|WSP>, <Wn|WSP>, #0

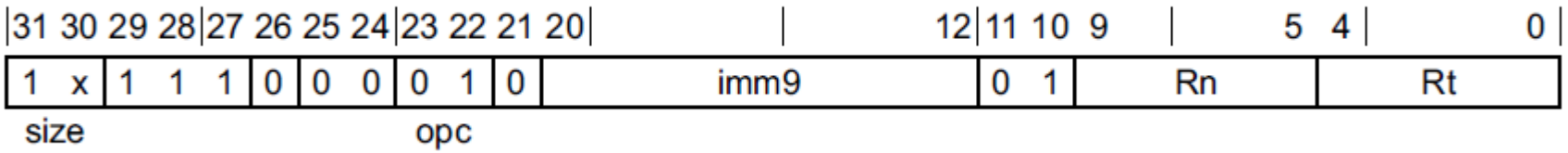
MOV <Wd>, #<imm>



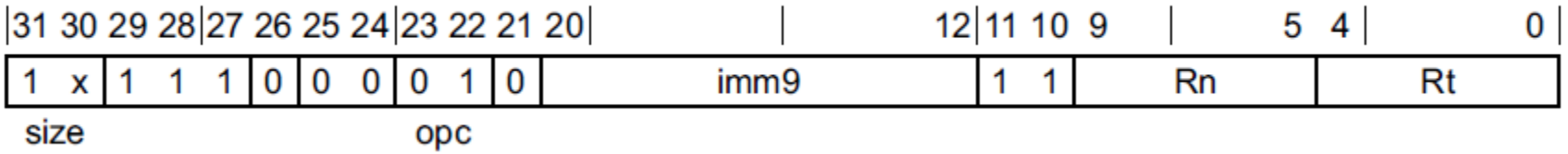

```
mov x1, #65535
mov x2, #65537
mov x3, #131070
```

arm v8a指令：LDR（立即数）

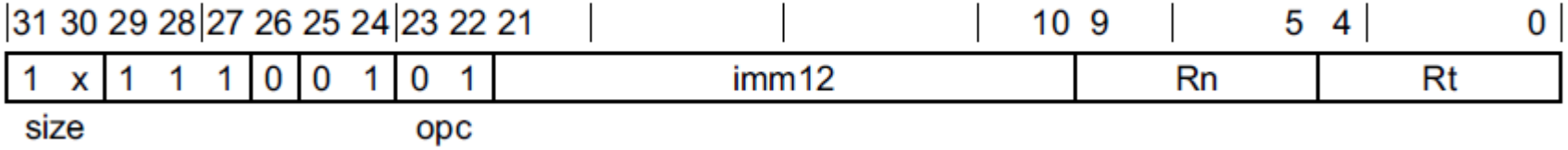
LDR <Xt>, [<Xn|SP>], #<simm>



LDR <Xt>, [<Xn|SP>, #<simm>]!



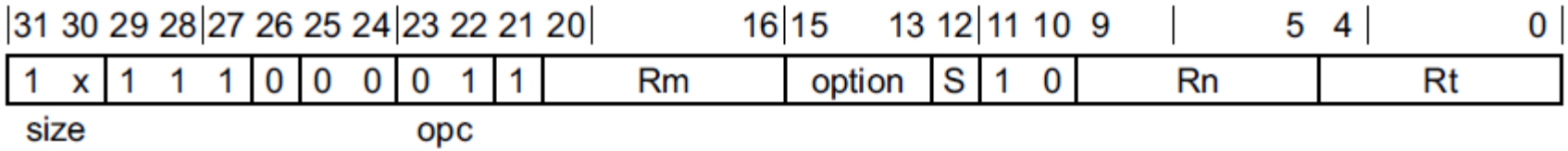
LDR <Xt>, [<Xn|SP>{, #<pimm>}]




```
ldr x2, [x1]
ldr x2, [x1, #10]
ldr x2, [x1, #10]!
```

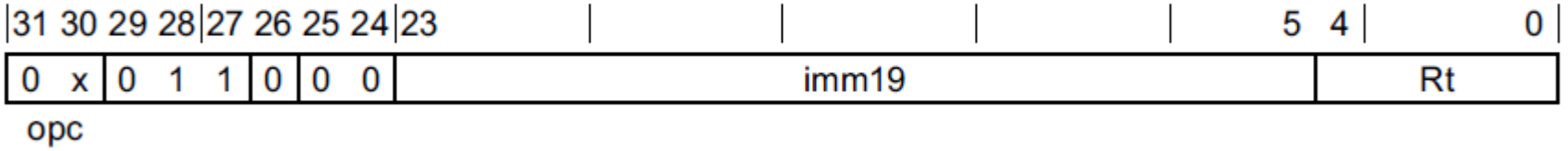
arm v8a指令：LDR（寄存器/标签）

```
LDR <Xt>, [<Xn|SP>, (<Wm>|<Xm>){, <extend> {<amount>}}]
```



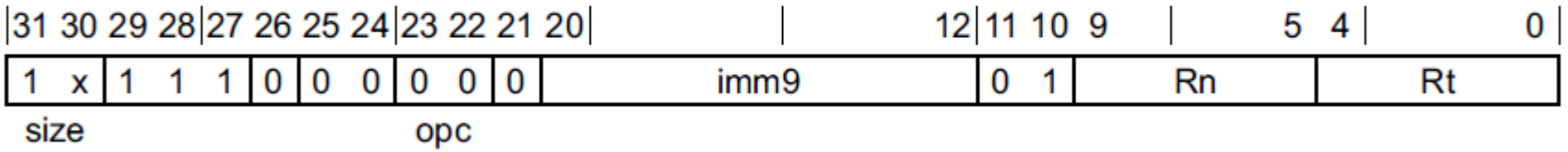

```
ldr x2, [x1, x0]
ldr x2, [x1, x0, lsl #3]
```

```
LDR <Xt>, <label>
```

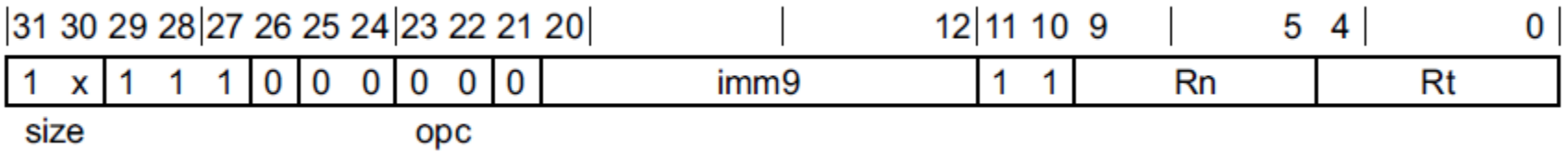


arm v8a指令：STR

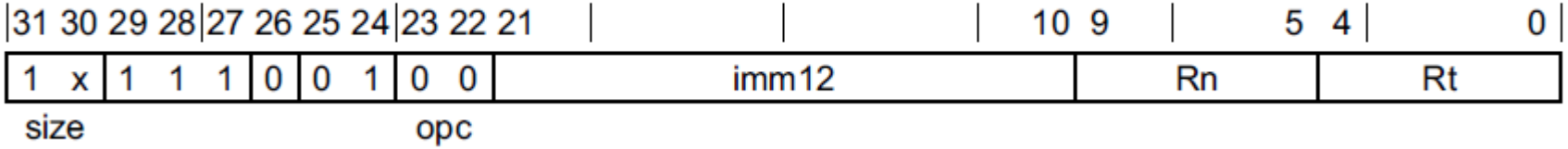
```
STR <Wt>, [<Xn|SP>], #<simm>
```



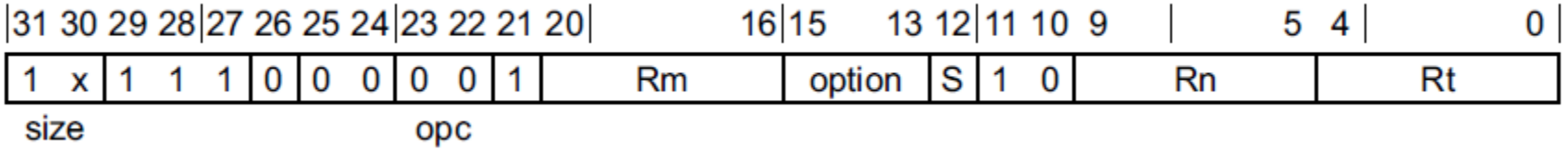

```
STR <Xt>, [<Xn|SP>, #<simm>]!
```




```
STR <Xt>, [<Xn|SP>{, #<pimm>}]
```




```
STR <Xt>, [<Xn|SP>, (<Wm>|<Xm>){, <extend> {<amount>}}]
```



复合类型如何翻译？

```
define i32 @main() {
  %1 = alloca [2 x i32]
  %2 = getelementptr [2 x i32], [2 x i32]* %1, i32 0, i32 0
  store i32 99, i32* %2
  %3 = load i32, i32* %2
  ret i32 %3
}
```

```
%mystruct = type { i32, i32 }
define i32 @main() {
  %1 = alloca %mystruct
  %2 = getelementptr %mystruct, %mystruct* %1, i32 0, i32 0
  store i32 1, i32* %2
  ret i32 0
}
```

IR=>Assembly：算数运算

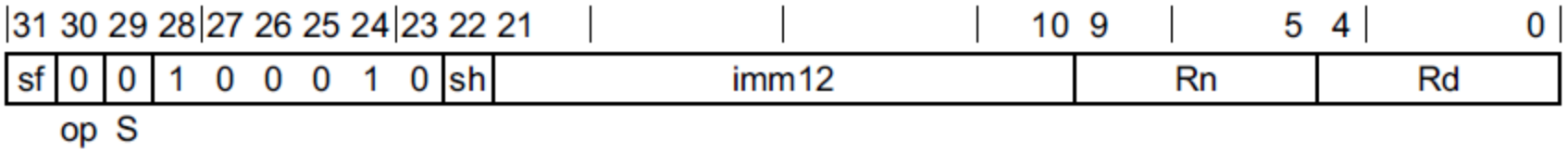
```
%r1 = add i32 %0, %1
%r2 = sub i32 %r1, 2
%r3 = mul i32 %r2, %1
%r4 = sdiv i32 %r3, %1
```

```
add  w8, w0, w1
sub  w8, w8, #2
mul  w8, w8, w1
sdiv w0, w8, w1
```

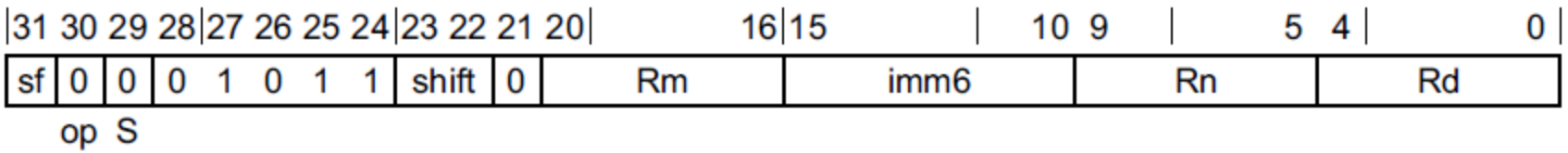
arm v8a指令：算数运算：ADD

```
ADD <Xd|SP>, <Xn|SP>, #<imm>{, <shift>}
```

sh == 1，imm左移12位




```
ADD <Xd>, <Xn>, <Xm>{, <shift> #<amount>}
```

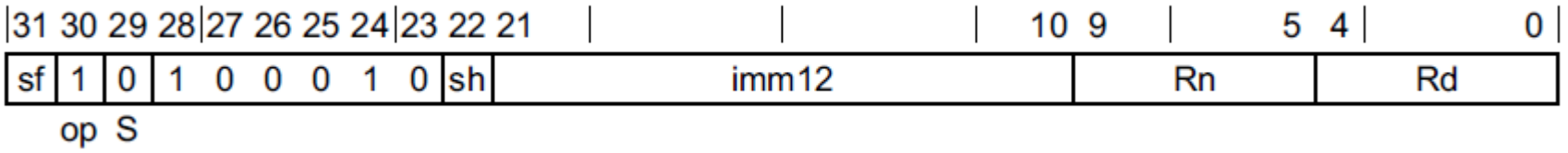



```
add x0, x1, #4095
add x0, x1, #4097
add x0, x1, #20480
```

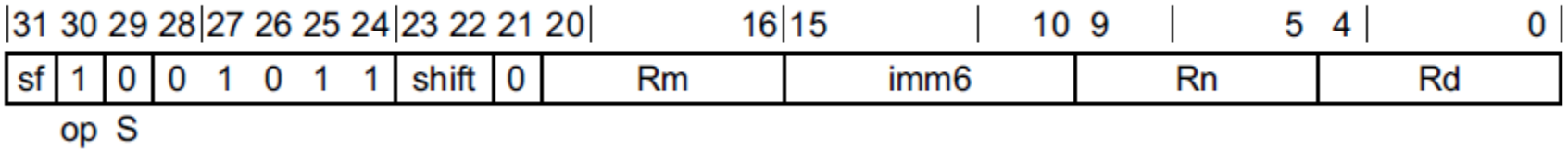
```
add x0, x1, x2
```

arm v8a指令：算数运算：SUB

```
SUB <Xd|SP>, <Xn|SP>, #<imm>{, <shift>}
```



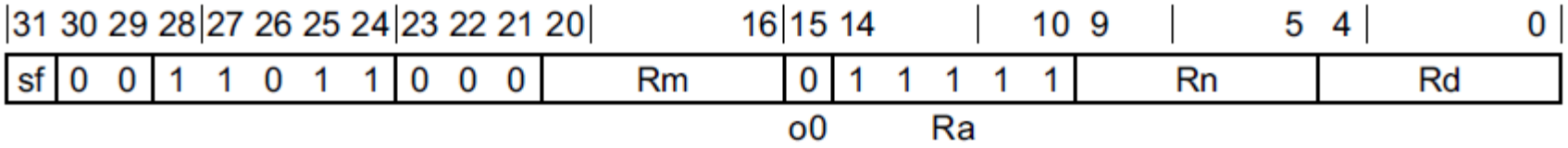

```
SUB <Xd>, <Xn>, <Xm>{, <shift> #<amount>}
```



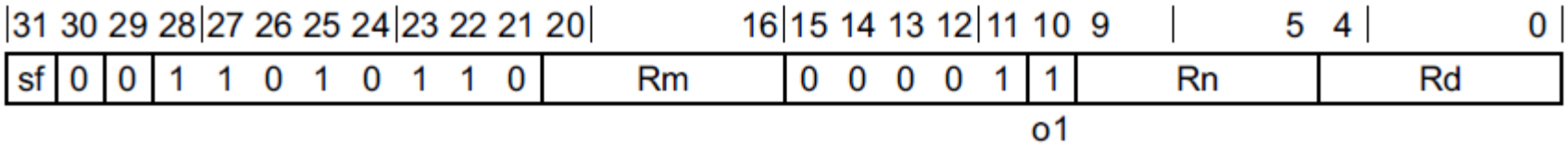
arm v8a指令：算数运算：MUL/SDIV

```
MUL <Xd>, <Xn>, <Xm>
```

不支持立即数

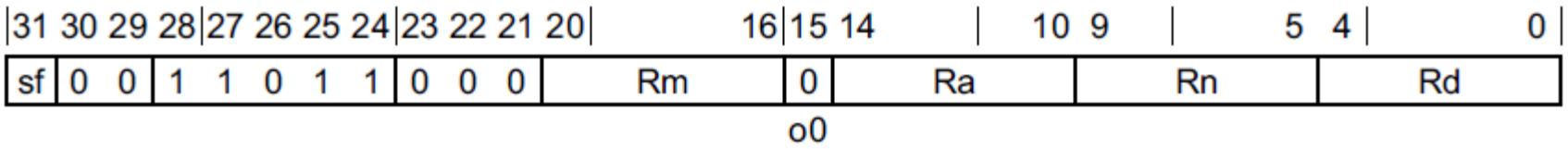



```
SDIV <Xd>, <Xn>, <Xm>
```

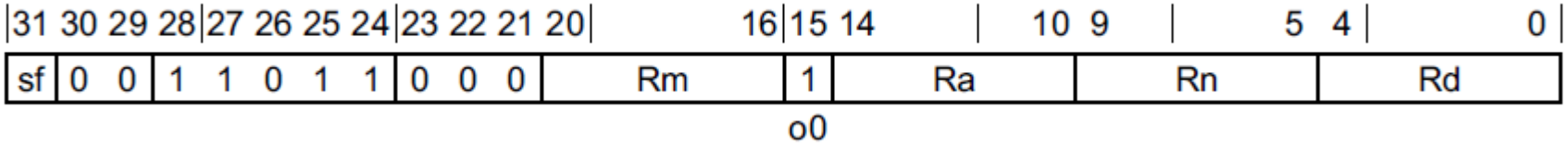


arm v8a指令：复合算数运算

MADD <Xd>, <Xn>, <Xm>, <Xa>



MSUB <Xd>, <Xn>, <Xm>, <Xa>




```
madd x0, x1, x2, x3
msub x0, x1, x2, x3
```

```
x0 = x1 * x2 + x3
x0 = x1 * x2 - x3
```

IR=>Assembly：比较运算和结果获取

```
%r1 = icmp sgt i32 %0, %1
%r2 = zext i1 %r1 to i32
```

```
cmp  w0, w1
cset w0, gt
```

条件

```
%r1 = icmp sge i32 %0, %1
%r2 = zext i1 %r1 to i32
```

```
cmp  w0, w1
cset w0, ge
```

```
%r1 = icmp eq i32 %0, %1
%r2 = zext i1 %r1 to i32
```

```
cmp  w0, w1
cset w0, eq
```

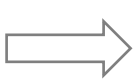
```
%r1 = icmp ne i32 %0, %1
%r2 = zext i1 %r1 to i32
```

```
cmp  w0, w1
cset w0, ne
```

```
%r1 = icmp sle i32 %0, %1
%r2 = zext i1 %r1 to i32
```

```
cmp  w0, w1
cset w0, le
```

```
%r1 = icmp lt i32 %0, %1
%r2 = zext i1 %r1 to i32
```

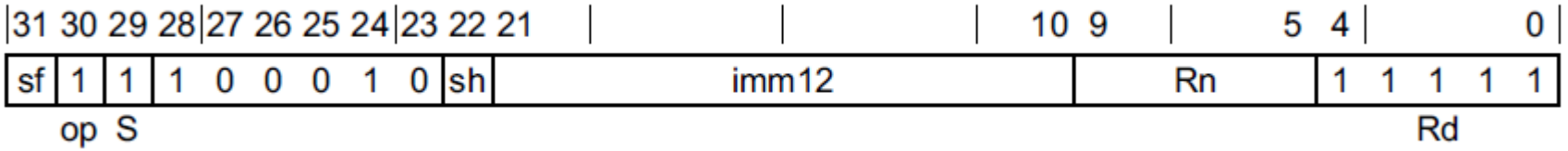


```
cmp  w0, w1
cset w0, lt
```

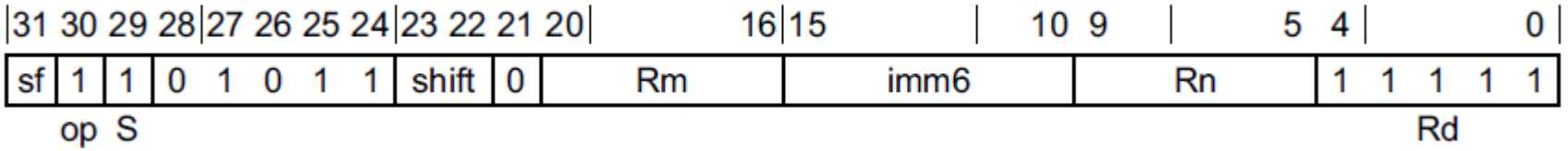
arm v8a指令：比较运算：CMP

- 基于PSR（NZCV）寄存器实现

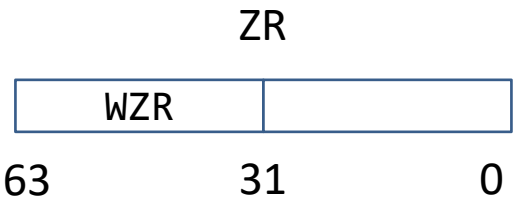
CMP <Xn|SP>, #<imm>{, <shift>}　　SUBS WZR, <Wn|WSP>, #<imm> {, <shift>}



CMP <Wn>, <Wm>{, <shift> #<amount>}



arm v8a寄存器：零寄存器




```
mov x1, xzr
mov w1, wzr
```

x1 = 0
w2 = 0

Arm v8a指令：读取NZCV

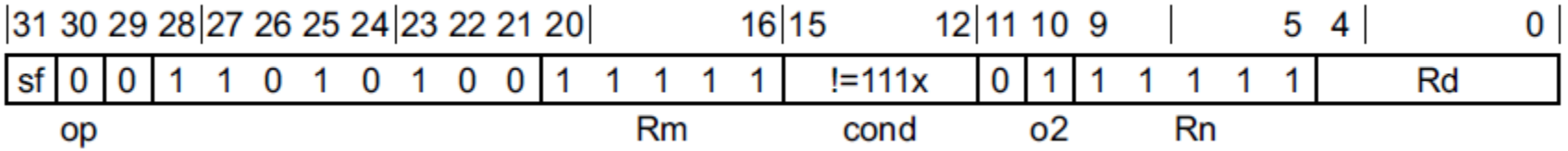
- 基于减法实现，更新CPSR寄存器：
  - N（31位）：符号标志位；如果负，则N=1
  - Z（30位）：0标志位；如果0，则Z=1
  - C（29位）：进位标志位；
    - 无符号数：加法进位，或减法不借位，则C=1
  - V（28位）：溢出标志位；有符号运算溢出，则V=1

Result	N	Z	C	V
Greater than	0	0	1	0
Less than	1	0	0	0
Equal	0	1	1	0

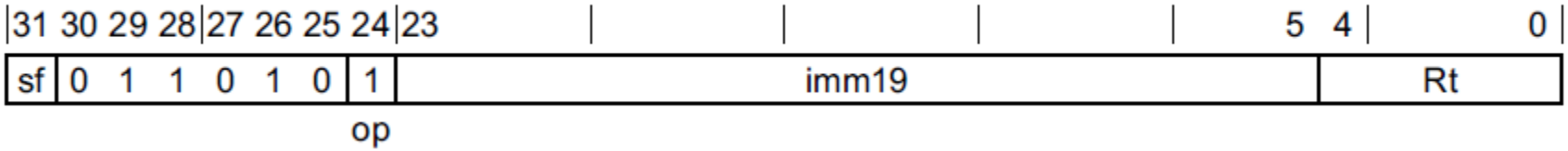
```
mrs x0, nzcv
```

Arm v8a指令：条件指令（举例）

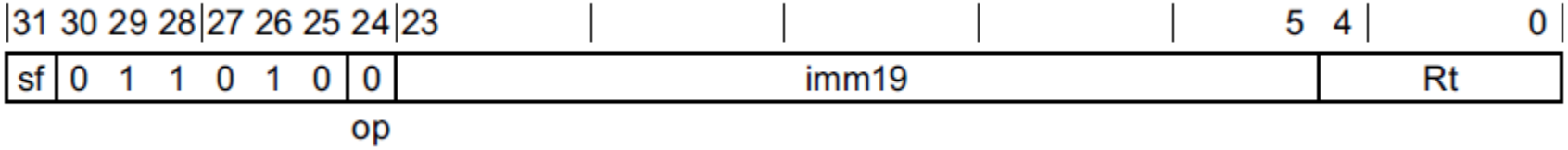
CSET <Xd>, <cond>



CBNZ <Xt>, <label>



CBZ <Xt>, <label>



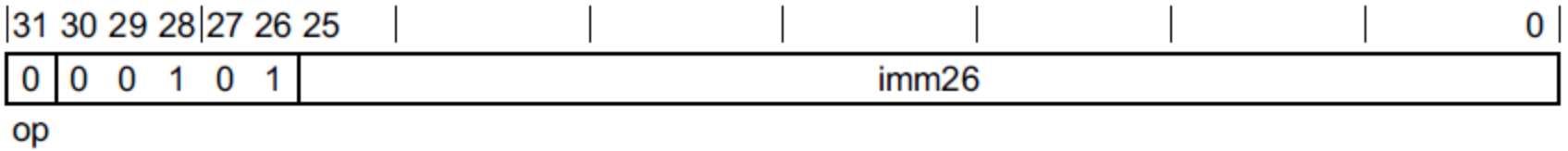
IR=>Assembly：跳转语句

```
bb0:
  %r1 = icmp sgt i32 %0, %1
  br i1 %r1, label %bb1, label %bb2
bb1:
  br label %bb3
bb2:
  br label %bb3
bb3:
```

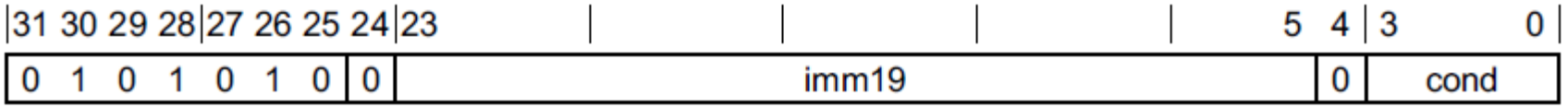
```
cmp     w0, w1
b.le    .LBB0_2
b       .LBB0_3
.LBB0_2:
.LBB0_3:
```

Arm v8a指令：跳转指令

B <label>



B.<cond> <label>



Mnemonic	Instruction	Branch offset range from the PC
B.cond	Branch conditionally	±1MB
BC.cond	Branch Consistent conditionally	±1MB
CBNZ	Compare and branch if nonzero	±1MB
CBZ	Compare and branch if zero	±1MB
TBNZ	Test bit and branch if nonzero	±32KB
TBZ	Test bit and branch if zero	±32KB

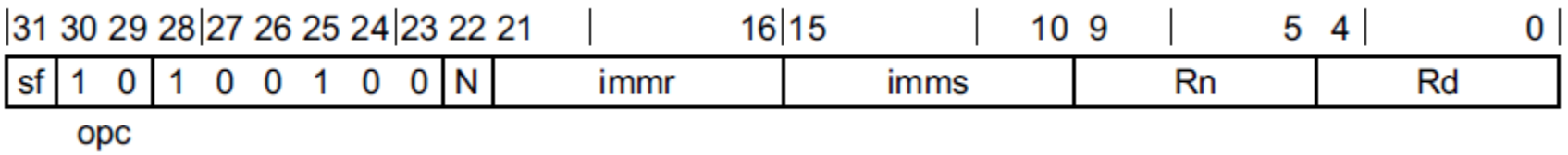
```
add w0, w1, w2
cbz zero_set
...
zero_set:
```

IR=>Assembly：异或运算（逻辑NOT）

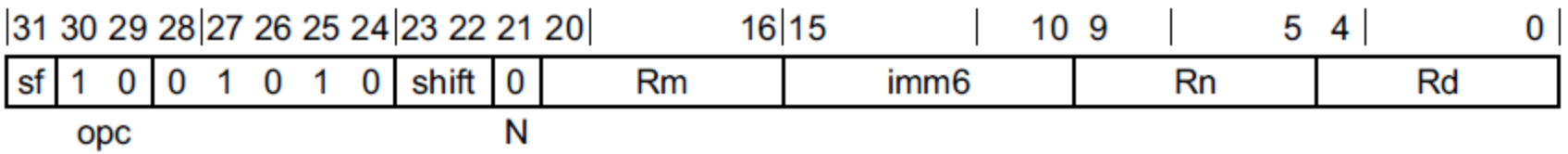
```
%r2 = xor i1 %r1, 0
```

```
eor  w0, w0, #1
```

EOR <Xd|SP>, <Xn>, #<imm>



EOR <Xd>, <Xn>, <Xm>{, <shift> #<amount>}



Mnemonic	Instruction
AND	Bitwise AND
ANDS	Bitwise AND and set flags
EOR	Bitwise exclusive OR
ORR	Bitwise inclusive OR
TST	Test bits

二、函数调用

IR=>Assembly：函数

```
@g = global i32 10

define i32 @foo(i32 %0) {
    %x = alloca i32
    store i32 %0, %x
    %g0 = load i32, i32* @g
    ret i32 %g0
}

define i32 @main() {
  %r0 = call i32 @foo(i32 1)
  ret i32 %r0;
}
```

```
foo:
    sub     sp, sp, #16
    adrp    x8, g
    add     x8, x8, :lo12:g
    str     w0, [sp, #12]
    ldr     w0, [x8]
    add     sp, sp, #16
    ret
main:
    str     x30, [sp, #-16]!
    mov     w0, #1
    bl      foo
    ldr     x30, [sp], #16
    ret
g:
    .word   10
```

1 word = 4 byte

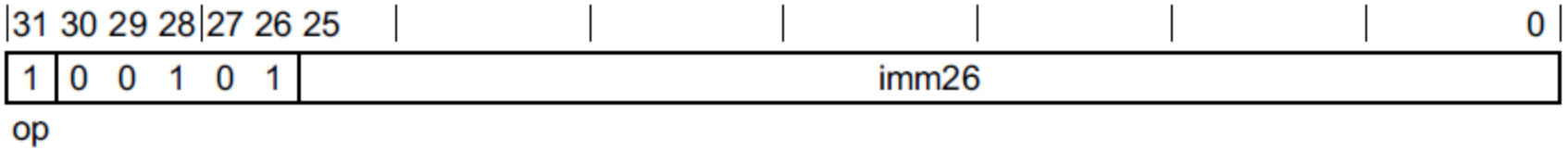
调用规约

- 参数传递: X0-X7
- 返回值: X0-X1
- Caller-saved Registers: X9-X15 （临时寄存器）
- Callee-saved Registers: X19-X28
- X29：一般用于栈帧基指针
- X30：一般用于返回地址
- SP：栈顶指针

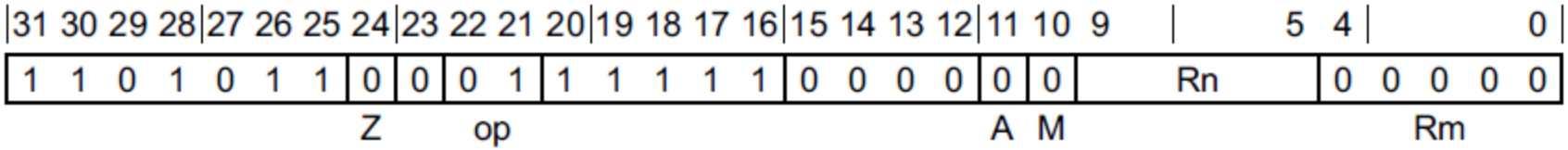
arm v8a指令：函数调用

- 跳转并将X30设置为PC+4

BL <label>



BLR <Xn>



IR=>Assembly：取址（全局变量）

Mnemonic	Instruction
ADRP	Compute address of 4KB page at a PC-relative offset
ADR	Compute address of label at a PC-relative offset.

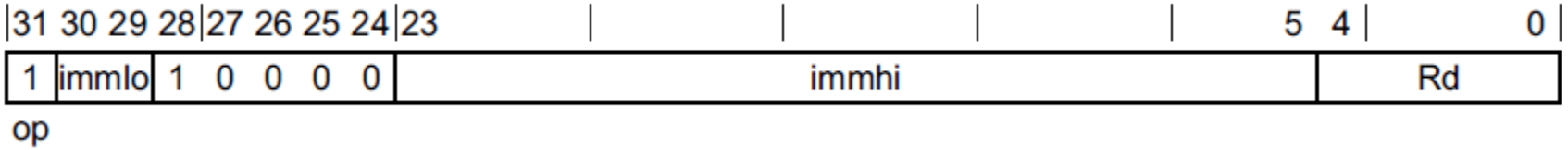
4KB对齐：末尾12位为0

```
adrp  x8, g
add   x8, x8, :lo12:g
```

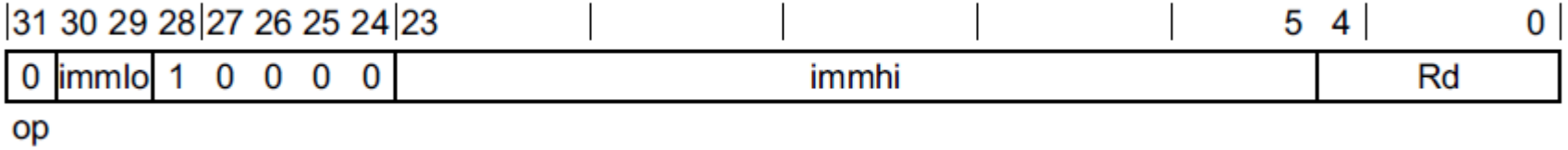
arm64版本：

```
adrp    x8, g@PAGE
add     x8, x8, g@PAGEOFF
```

ADRP <Xd>, <label>



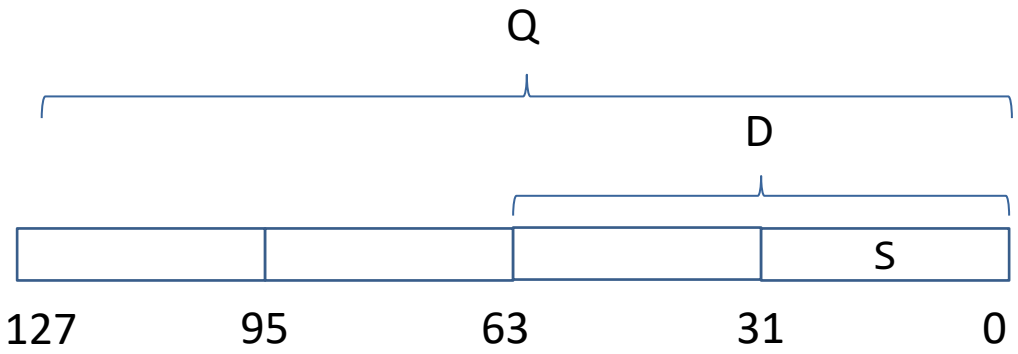
ADR <Xd>, <label>



三、更多指令和寄存器

arm v8a寄存器：向量寄存器和SIMD指令

- 向量寄存器：Q0-Q31




```
VLDR S0, [X0]
VLDR S1, [X1]
VADD.F32 S2, S0, S1
```

浮点数运算

```
VLDR Q0, [X0]
VLDR Q1, [X1]
VADD.I32 Q2, Q0, Q1
```

向量运算

参考资料

- Arm® Architecture Reference Manual for A-profile architecture
- 在线模拟器：http://163.238.35.161/~zhangs/arm64simulator/