

2-3

此部分主要讨论溢出检测，是本章的重点。

(1) $hsp \rightarrow addi x0, x0, 0$

(2) $ret \rightarrow jalr x0, 0(x1)$ 通过返回地址 TLR 中来启动函数 V-JMP

(3) $call offset \rightarrow auipc x6, offset[31:12]$ 在分支前设置跳转点；

通过时钟脉冲，依次是：jalr, $x1, x6, offset[11:0]$ 同样地，在分支后

(4) $mv rd, rs \rightarrow addi rd, rs, 0$ 将各条语句的相对地址写入周期

(5) $rdcycle rd \rightarrow csrrs rd, cycle, x0$

(6) $sext.w rd, rs \rightarrow addiw rd, rs, 0$

已知固定数

子函数 END

2-7

举列说明直接寻址、WZR/WZR、WZR/WZR、WZR/WZR

(1) $addi t0, t1, t2$ WZR/WZR、WZR/WZR、WZR/WZR

$sub t3, t0, t1$

$mv t4, t2$

$bne t3, t4, overflow$

(2) 对于无符号数加法：

$addu t0, t1, t2$

$bltu t0, t1, overflow$

(3) ARM 指令集通过 CPSR 状态寄存器表示溢出：(QPSR, N, C, V, C, V)

无符号数加法溢出时（进位）将 C 寄存器置 1，减法溢出时（借位）将 C 寄存器置 0。

有符号数：有符号数加减法中，若发生溢出，将 V 寄存器置 1

$x6$ 通过 CF 和 OF 寄存器来检测溢出：

无符号数：无符号数加减法出现溢出时，将 CF 寄存器置 1

有符号数：有符号数加减法出现溢出时，将 OF 寄存器置 1



扫描全能王 创建

2-8

(1) 指令

Op rd,rs1,rs2 x 0 $-2^{k-1} \leq x \leq 1$ -1 0

除数为0时不会抛出异常，而是产生某个特殊的默认值，同时设置某些状态寄存器的状态位，因为这样可以大幅简化处理器流水线的硬件实现。

(2) fflags 各位的含义：

NV : 无效操作

DZ : 除以零

OF : 溢出

UF : 下溢

NX : 不精确

fflags 被置位不会使处理器陷入系统调用。

(3) x86 架构：CPU 通过 8 位的中断类型码，通过中断向量表找到对应的中断处理程序的入口地址，随即控制权交由操作系统内核进行故障处理。

ARM 架构：同样会抛出一个异常，由异常处理程序处理。



扫描全能王 创建

2-12.

(1) Linux Kernel UML 模式 UML 2011-09 立的 11

(2) Boot ROM 1- M 模式 1-2011-09-26 2011-09

(3) Boot Loader S 模式

(4) USB Driver 驱动层 S 模式

(5) vim 编辑器驱动层 U 模式

2-13.

寄存器表： VN

start : addi sp, sp, -48

重压表： VD

sw \$0, 44(sp)

压表： VD

addi \$0, sp, 48

压表： VN

sw \$0, -36(\$0) # A 压表： VN

sw \$1, -40(\$0) # B 压表： VN

将值从内存中读出并写入寄存器，通过直接寻址，将值从内存中读出并写入寄存器

part1: sw \$0, -20(\$0) # 将值从内存中读出并写入寄存器，通过间接寻址

j part4 # part4 函数部分由本程序下部分完成，即从 MRA

part2: lw a5, -20(\$0) # a5: i

slli a5, a5, 2 # 左移 2 位，即给 i 乘 4，因为数组中一个 int 占 4 字节

lw a4, -40(\$0) # a4: B

add a4, a4, a5

lw a6, 0(a4) # a6: Bi[1]

lw a3, -36(\$0) # a3: A

add a3, a3, a5

lw a7, -44(\$0) # a7: C



扫描全能王 创建

mul ab. ab. a7 # B7i] * C

sw ab, 0(a3) # 有入 A[i]

part3: lw a5, -20(\$0)

addi a5. a5. 1 # i++

sw a5, -20(\$0)

part4: lw a5, -20(\$0)

li a4, 99

ble a5, a4, part2 # i < 99 时循环

end: lw a5, -36(\$0)

lw a5, 0(a5) # return A[0]

mr a0, a5

lw \$0, 44(\$p)

cddi sp, \$p - 48

jr ra

2-14

start: addi \$p - \$p, -32 # (0) 00. 20 wl

sw \$0, 24(\$p) # 0. 20 ibba

addi \$0, \$p = 32 # (0) 00. 20 wl

sw a0, -20(\$0) # 9 (0) 00. 20 wl

sw a1, -24(\$0) # b. 00. 20 wl

sw a2, -28(\$0) # c. 00. 20 wl

part1: lw a5, -20(\$0)

lw a4, -24(\$0)



扫描全能王 创建

ble as, a4, part2 # a < b 时, Bif 取 else w1

add a3, as, a4 # (a1) C, a3, a4, w1

sw a3, -28(\$0) (02) a6+, 2a, w1 : 371, w1

j end # 1, 2a, 2a, 11ba

part2: sub a3, as, a4 # (a1) 05+, 2a, w2

sw a3, -28(\$0) (02) a6+, 2a, w1 : 4-371

end: lw \$0, 24(\$sp) pp, +a, w1

addi sp, \$p, 32 # 4-371, +a, 2a, 91d

jr ra . (02) a6-, 2a, w1 : 2-371

(02) A MINUS # (2A) 0, 2a, w1

2-15. 2a, 0a, w1

start: addi sp, \$p, -32 (02) pp, 02, w1

sw \$0, 24(\$p) sp, q2, q2, 11ba

addi \$0, \$p, 32 # 11ba, w1

sw t0, -20(\$0) # p

sw t1, -24(\$0) # a

part1: lw a5, -20(\$0) # a5: p, -q2, q2, 11ba : 3-372

addi a4, a5, 0 # a4 & p[5] 02, w1

sw a5, 0(a4) # p[5] = p, 02, 11ba

part2: lw a5, -24(\$0) # (a5: p, -q2, q2, 11ba, w1)

li a4, 3 # (R) +2+ , 1a, w2

sw a4, -24(\$0) # ((a=3, -1a, w2

part3: lw a5, -20(\$0) # (a5: p, -2a, w1 : 1-372

lw a4, -24(\$0) # (a4 + a, +a, w1



addi a3, a3, -4 # a3:&p[1] w
sw a4, 0(a3) # p[1]=a. w
part 1: lw a5, -20(\$0) # a5:p. w
lw a4, -24(\$0) # a4:a. w
slli a6, a4, 2 # a6: a×4. w
add a7, a5, a6 # a7: &p[a]. w
sw a4, 0(a7) # p[a]=a
end: lw \$0, -24(\$0)
addi = 10 sp, sp, -32
jr ra

2-16
start addi sp, sp, -32
sw \$0, -24(sp) (++(\$0) : EA > UE do) rot

addi \$0, sp, 32, L*10 = 10

sw t0, -20(\$0) # a

sw t1, -24(\$0) # b

part 1: lw a5, -20(\$0)

lw a4, 0(a5) # a4: tmp = *a

sw a4, -28(\$0)

part 2: lw a5, -24(\$0) # a5:b

lw a4, 0(a5) # a4: *b

lw a3, -20(\$0) # a3:a

sw a4, 0(03) # *a = *b



扫描全能王 创建

part 3: lw \$5. -24(\$0) # \$5: bin. 60 ibn
 lw \$4. -28(\$0) # \$4: tmp. 40 we
 sw \$4. 0(\$5) # \$4:tmp. 28 wl : bin
 end: lw \$0, 24(\$p) # (\$0) 0x00. 40 wl
 addi \$p, \$p, 32 # 0. 40. 00 bin
 jr \$ra, \$p # return. 50 ibn
 N = \$0.0 # (\$0) 0. 40 we

2-17

代码功能：让 a1 的值乘以 2^{30} ，即循环 30 次 $a1 = a1 \times 2$

转换为 C 语言：

int a0 = 0

int a1 = 1

int a2 = 30

for (; a0 < a2 ; a0++)

$a1 = a1 * 2$

N = (\$0) 00. 00 we

d := (\$0) 00. 10 we

(00) 00. 70 wl : 1000

$n^k = d * t : 40$ # (20) 0. 40 wl

(00) 00. 40 we

d := 20 # (00) 00. 20 wl : 1000

d := 20 # (20) 0. 20 wl

N := 0 # (00) 00. 00 wl

d := 1 # (00) 0. 00 we



扫描全能王 创建