目录

[1 概述 2](#_Toc504840847)

[1.1 指令集 2](#_Toc504840848)

[1.2 伪指令 2](#_Toc504840849)

[1.3 指令集插件开发和伪指令插件开发 2](#_Toc504840850)

[2 指令集 3](#_Toc504840851)

[2.1 通用指令集插件 3](#_Toc504840852)

[2.1.1 show 3](#_Toc504840853)

[2.1.2 showstring 3](#_Toc504840854)

[2.1.3 add 3](#_Toc504840855)

[2.1.4 equ 4](#_Toc504840856)

[2.1.5 gt 4](#_Toc504840857)

[2.1.6 gte 4](#_Toc504840858)

[2.1.7 lt 5](#_Toc504840859)

[2.1.8 lte 5](#_Toc504840860)

[2.1.9 mul 5](#_Toc504840861)

[2.1.10 sub 6](#_Toc504840862)

[2.1.11 bjmp 6](#_Toc504840863)

[2.1.12 cjmp 6](#_Toc504840864)

[2.1.13 end 7](#_Toc504840865)

[2.1.14 halt 7](#_Toc504840866)

[2.1.15 jmp 7](#_Toc504840867)

[2.1.16 ldr 8](#_Toc504840868)

[2.1.17 ldrb 8](#_Toc504840869)

[2.1.18 mov 9](#_Toc504840870)

[2.1.19 pop 9](#_Toc504840871)

[2.1.20 push 9](#_Toc504840872)

[2.1.21 ret 10](#_Toc504840873)

[2.1.22 str 10](#_Toc504840874)

[2.1.23 strb 10](#_Toc504840875)

[2.2 程序域指令集插件 11](#_Toc504840876)

[2.2.1 fork 11](#_Toc504840877)

[3 伪指令 11](#_Toc504840878)

[3.1 通用伪指令 11](#_Toc504840879)

[3.1.1 di\_strstring 11](#_Toc504840880)

[4 指令集插件开发 12](#_Toc504840881)

[5 伪指令插件开发 12](#_Toc504840882)

# 概述

## 指令集

assembly-n指令采用三段制，格式如下：

***instruction [arg1],[arg2],[arg3]***

参数从0-3个不等

1、assembly-n通用指令集插件定义了22个指令，分别是

|  |  |
| --- | --- |
| 调试指令 | show、showstring |
| 数学指令 | add、equ、gt、gte、lt、lte、mul、sub |
| 基础指令 | bjmp、cjmp、end、halt、jmp、ldr、ldrb、mov、pop、push、ret、str、strb |

2、assembly-n域指令集插件定义了1个指令：fork

## 伪指令

assembly-n伪指令没有特定形式。

1、通用伪代码指令插件

|  |  |
| --- | --- |
| 基础指令 | strstring |

## 指令集插件开发和伪指令插件开发

为了便于扩展assembly-n，所有的指令和伪指令都是以插件的形式出现的。指令集插件和伪指令都需要实现相应的接口才能被加载到核心中。所有的插件均需要C#或其他的CLR平台语言开发。

如何知道需要开发的功能使用那种插件开发？

首先需要了解一下指令集和伪指令的区别。指令集是在虚拟机运行时才会执行的指令；伪指令在伪指令编译时会被编译成指令，在运行过程中是没有伪指令概念的。所谓的指令集插件开发是对虚拟机功能的扩展，它所做的操作也只能是对虚拟机资源进行操作。伪指令插件开发则是对指令集的进一步抽象，也就是一个目标语言是assembly-n的一个编译器。

第二，指令集能操作什么。程序集能操作的东西有：1、寄存器，总共15个寄存器。2、内存，由于地址偏移的存在，指令集只能操作除去代码区外的内存。3、参数，到了运行时这个状态，代码的概念已经不存在了，指令集可以处理三个编译后的参数，这三个参数都是32位的数据。

第三，伪指令能操作什么。伪指令能操作的东西只有源代码，但是这个是最原始的源代码。所以很多虚拟机操作起来很复杂的东西就可以通过伪指令实现了。比如字符串记载，由于虚拟机不支持字符作为参数，所以必须将字符串每个字符转化为整数类型后依次写入内存，这个过程实际上可以通过伪指令来实现。

# 指令集

## 通用指令集插件

### show

|  |  |
| --- | --- |
| 作用 | 在屏幕上打印一个寄存器或数字 |
| 参数1 | 需要打印的对象，寄存器或数字 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### showstring

|  |  |
| --- | --- |
| 作用 | 在屏幕上打印字符串 |
| 参数1 | 需要打印的对象地址。可以是寄存器或数字 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### add

|  |  |
| --- | --- |
| 作用 | 加法指令,结果arg1=arg2+arg3 |
| 参数1 | 加法结果存储位置，必须是寄存器 |
| 参数2 | 加数，必须是寄存器 |
| 参数3 | 被加数，必须是寄存器 |

### equ

|  |  |
| --- | --- |
| 作用 | 相等判断指令，如果相等cpsr寄存器溢出位置位 |
| 参数1 | 数1，必须是寄存器 |
| 参数2 | 数2，必须是寄存器 |
| 参数3 | 无效 |

### gt

|  |  |
| --- | --- |
| 作用 | 大于判断指令，如果大于cpsr寄存器溢出位置位 |
| 参数1 | 数1，必须是寄存器 |
| 参数2 | 数2，必须是寄存器 |
| 参数3 | 无效 |

### gte

|  |  |
| --- | --- |
| 作用 | 大于等于判断指令，如果大于等于cpsr寄存器溢出位置位 |
| 参数1 | 数1，必须是寄存器 |
| 参数2 | 数2，必须是寄存器 |
| 参数3 | 无效 |

### lt

|  |  |
| --- | --- |
| 作用 | 小于判断指令，如果小于cpsr寄存器溢出位置位 |
| 参数1 | 数1，必须是寄存器 |
| 参数2 | 数2，必须是寄存器 |
| 参数3 | 无效 |

### lte

|  |  |
| --- | --- |
| 作用 | 小于等于判断指令，如果小于等于cpsr寄存器溢出位置位 |
| 参数1 | 数1，必须是寄存器 |
| 参数2 | 数2，必须是寄存器 |
| 参数3 | 无效 |

### mul

|  |  |
| --- | --- |
| 作用 | 乘法指令，结果arg1=arg2\*arg3 |
| 参数1 | 结果存储位置，必须是寄存器 |
| 参数2 | 被乘数，必须是寄存器 |
| 参数3 | 乘数，必须是寄存器 |

### sub

|  |  |
| --- | --- |
| 作用 | 减法指令，结果arg1=arg2-arg3 |
| 参数1 | 结果存储位置，必须是寄存器 |
| 参数2 | 被减数，必须是寄存器 |
| 参数3 | 减数，必须是寄存器 |

### bjmp

|  |  |
| --- | --- |
| 作用 | 带返回的跳转，与ret配合使用实现函数功能 |
| 参数1 | 跳转位置，section名或存储section地址的寄存器 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### cjmp

|  |  |
| --- | --- |
| 作用 | 带条件的跳转，cpsr溢出位为1跳转，否则执行下一条指令 |
| 参数1 | 跳转位置，section名或存储section地址的寄存器 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### end

|  |  |
| --- | --- |
| 作用 | 程序结束指令，每个程序结束位置必须写此指令 |
| 参数1 | 无效 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### halt

|  |  |
| --- | --- |
| 作用 | 休眠指令，休眠1ms |
| 参数1 | 无效 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### jmp

|  |  |
| --- | --- |
| 作用 | 休眠指无条件跳转指令 |
| 参数1 | 跳转位置，section名或存储section地址的寄存器 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### ldr

|  |  |
| --- | --- |
| 作用 | 数据加载指令，从内存中加载数据到寄存器 |
| 参数1 | 数据加载到的寄存器，必须是寄存器 |
| 参数2 | 数据地址，从这个地址读取数据，寄存器或立即数 |
| 参数3 | 无效 |

### ldrb

|  |  |
| --- | --- |
| 作用 | byte数据加载指令，从内存中加载1byte数据到寄存器 |
| 参数1 | 数据加载到的寄存器，必须是寄存器 |
| 参数2 | 数据地址，从这个地址读取数据，寄存器或立即数 |
| 参数3 | 偏移地址，一个地址有4个byte，此参数指定数据从哪个byte加载 |

### mov

|  |  |
| --- | --- |
| 作用 | 数据转移指令 |
| 参数1 | 目标寄存器，必须是寄存器 |
| 参数2 | 来源数据，寄存器或立即数 |
| 参数3 | 无效 |

### pop

|  |  |
| --- | --- |
| 作用 | 出栈操作 |
| 参数1 | 出栈后的数据存储位置，必须是寄存器 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### push

|  |  |
| --- | --- |
| 作用 | 入栈操作 |
| 参数1 | 需要入栈的数据位置，必须是寄存器 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### ret

|  |  |
| --- | --- |
| 作用 | 返回指令，与bjmp配合实现函数功能 |
| 参数1 | 无效 |
| 参数2 | 无效 |
| 参数3 | 无效 |

### str

|  |  |
| --- | --- |
| 作用 | 存储数据到内存 |
| 参数1 | 需要存储的数据，必须是寄存器 |
| 参数2 | 存储数据的地址，寄存器或立即数 |
| 参数3 | 无效 |

### strb

|  |  |
| --- | --- |
| 作用 | 存储1byte数据到内存 |
| 参数1 | 需要存储的数据，必须是寄存器 |
| 参数2 | 存储数据的地址，寄存器或立即数 |
| 参数3 | 偏移地址，一个地址有4个byte，此参数指定数据存储到哪个byte |

## 程序域指令集插件

### fork

|  |  |
| --- | --- |
| 作用 | 分支指令，从该条指令开始会从当前虚拟机拷贝一个新的虚拟机，同时运行两个虚拟机。如果是新创建的虚拟机，会跳转到参数1指定的位置继续运行，否则运行下一条指令。 |
| 参数1 | 新虚拟机跳转的位置 |
| 参数2 | 无效 |
| 参数3 | 无效 |

# 伪指令

## 通用伪指令

### di\_strstring

存储一个字符串到指定地址的内存中。

该指令有两个参数：

参数1：字符串存储的地址，必须是寄存器。

参数2：需要存储的字符串，由双引号引住。

示例（将字符串存储到地址20开始的内存中 ）：

mov r0,20

di\_strstring r0,”Hello assembly-n”

# 指令集插件开发

指令集插件开发首先需要引用asn.Runtime.Interface.dll。创建一个指令集处理类，继承实现接口IOperator。虚拟机会给指令处理方法传递三个参数：

Runtime参数：运行时，是一个操作虚拟机的接口。

Args参数：指令的参数，最多三个。

Types参数：每个指令的类型。MemoryAttribute枚举中定义了参数类型的含义

# 伪指令插件开发

伪指令插件开发也需要引用asn.Runtime.Interface.dll。创建一个伪指令处理类，继承实现接口IDummyInstructions。主要需要实现接口的两个方法Input和Result。Input方法处理源代码，Result返回处理结果。

编译器会给Input方法传递三个参数：

line：源代码行，原始的源代码行，需要根据自己的功能自行解析。

codes：所有的源代码行。

index：当前源代码行在codes中的索引位置。

编译器会Result获取编译结果，编译结果是一个字符串的列表，结果内只能包含指令集的指令。