Capstone-Engine API Documentation

V4.0.2

Official API document by kabeor

Translated by Nitr0-G.

Capstone Engine is a binary disassembly engine that supports multiple hardware architectures.

0x0 development preparation

Capstone official website: http://www.capstone-engine.org

Compile lib and dll methods by yourself
Source code: https://github.com/capstone-engine/capstone.git

The file structure of git clone is as follows：

```
.                <- 主要引擎core engine + README + 编译文档COMPILE.TXT 等
├── arch         <- 各语言反编译支持的代码实现
│   ├── AArch64  <- ARM64 (aka ARMv8) 引擎
│   ├── ARM      <- ARM 引擎
│   ├── EVM      <- Ethereum 引擎
│   ├── M680X    <- M680X 引擎
│   ├── M68K     <- M68K 引擎
│   ├── MOS65XX  <- MOS65XX 引擎
│   ├── Mips     <- Mips 引擎
│   ├── PowerPC  <- PowerPC 引擎
│   ├── Sparc    <- Sparc 引擎
│   ├── SystemZ  <- SystemZ 引擎
│   ├── TMS320C64x <- TMS320C64x 引擎
│   ├── X86      <- X86 引擎
│   └── XCore    <- XCore 引擎
├── bindings     <- 绑定
│   ├── java     <- Java 绑定 + 测试代码
│   ├── ocaml    <- Ocaml 绑定 + 测试代码
│   ├── powershell <- powershell 绑定 + 测试代码
│   ├── python   <- python 绑定 + 测试代码
│   └── vb6      <- vb6 绑定 + 测试代码
├── contrib      <- 社区代码
├── cstool       <- Cstool 检测工具源码
├── docs         <- 文档，主要是capstone的实现思路
├── include      <- C头文件
├── msvc         <- Microsoft Visual Studio 支持 (Windows)
├── packages     <- Linux/OSX/BSD包
├── suite        <- 项目开发所需工具
├── tests        <- C语言测试用例
├── windows      <- Windows 支持(Windows内核驱动编译)
├── windowsce    <- Windows CE 支持
└── xcode        <- Xcode 支持 (MacOSX 编译)
```

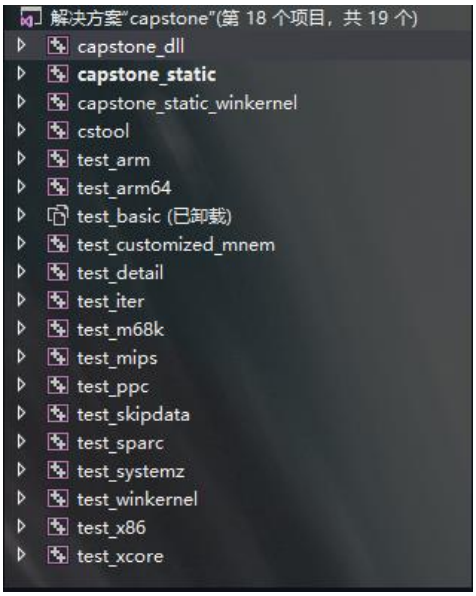
The following demonstrates that Windows10 is compiled using Visual Studio2019

Copy the msvc folder to a more refreshing location, the internal structure is as follows ：

capstone_dll	2019/7/19 13:06	文件夹	
capstone_static	2019/7/19 13:12	文件夹	
capstone_static_winkernel	2019/1/10 21:45	文件夹	
cstool	2019/7/19 13:06	文件夹	
Debug	2019/7/19 13:12	文件夹	
test_arm	2019/7/19 13:06	文件夹	
test_arm64	2019/7/19 13:06	文件夹	
test_customized_mnem	2019/7/19 13:06	文件夹	
test_detail	2019/7/19 13:06	文件夹	
test_iter	2019/7/19 13:06	文件夹	
test_m68k	2019/7/19 13:06	文件夹	
test_mips	2019/7/19 13:06	文件夹	
test_ppc	2019/7/19 13:06	文件夹	
test_skipdata	2019/7/19 13:06	文件夹	
test_sparc	2019/7/19 13:06	文件夹	
test_systemz	2019/7/19 13:06	文件夹	
test_winkernel	2019/1/10 21:45	文件夹	
test_x86	2019/7/19 13:06	文件夹	
test_xcore	2019/7/19 13:06	文件夹	
capstone.sln	2019/7/19 13:12	Visual Studio Sol...	16 KB
README	2019/1/10 21:45	文件	2 KB

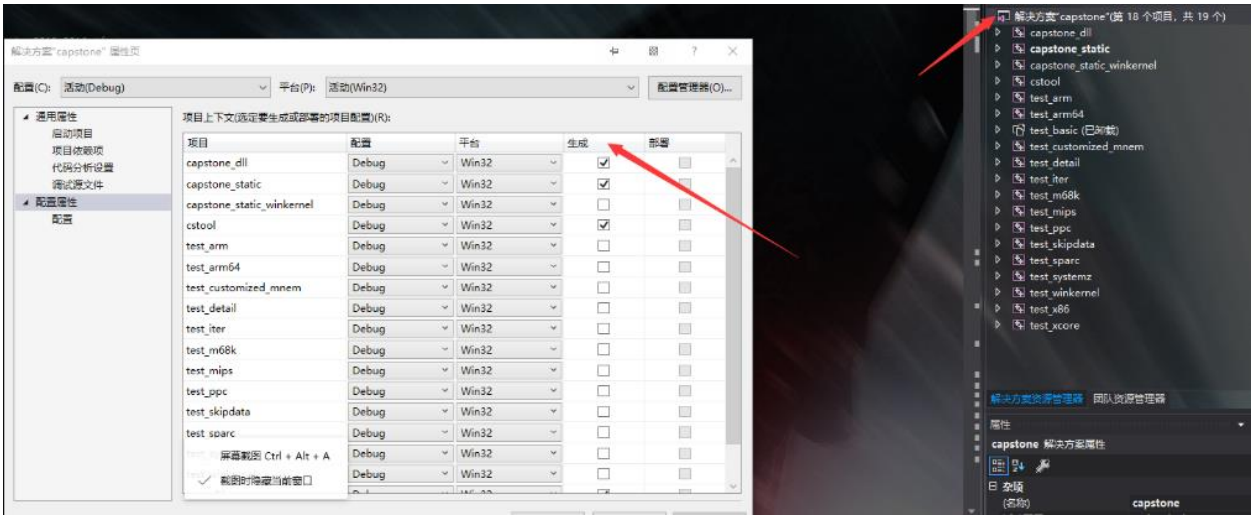
;EU - VS open capstone.sln project files, the solution automatically loads these:

;RU – Откройте в VS capstone.sln и solution автоматически загрузит этоЖ



;EU - You can see that all the supported languages are here. If you need them all, just compile them directly. Only a few of them are needed. Then right-click solution->Properties->Configure the properties as follows:

;RU - Вы можете видеть, что здесь представлены все поддерживаемые языки. Если они вам нужны все, просто скомпилируйте их напрямую. Нужны лишь некоторые из них. Затем щелкните правой кнопкой мыши solution->Properties->Настройте свойства следующим образом:



;EU - Check the support items you need in the build options to compile and capstone will be generated in the Debug directory of the current folder.lib static compilation library and capstone.dll dynamic library so that you can start using Capstone for development.

;RU - Проверьте необходимые элементы поддержки в параметрах сборки для компиляции, и capstone будет сгенерирован в каталоге Debug текущей folder.lib статической компиляции lib и capstone.dll, чтобы вы могли начать использовать Capstone для разработки.

If you don't want to compile by yourself, the official compiled version is also provided

Win32： https://github.com/capstone-engine/capstone/releases/download/4.0.2/capstone-4.0.2-win32.zip

Win64： https://github.com/capstone-engine/capstone/releases/download/4.0.2/capstone-4.0.2-win64.zip

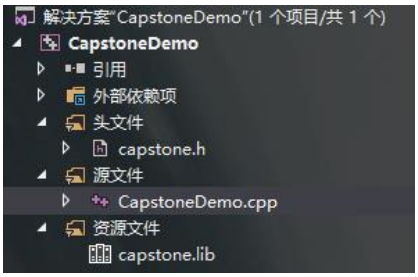
Choosing x32 or x64 will affect the number of digits developed later

Engine call test

Create a new VS project, copy all the header files in capstone\include\capstone and the compiled lib and dll files to the home directory of the new project

.vs	2019/7/19 13:17	文件夹	
Debug	2019/7/19 14:49	文件夹	
x64	2019/7/19 14:51	文件夹	
arm.h	2019/1/10 21:45	C/C++ Header	19 KB
arm64.h	2019/1/10 21:45	C/C++ Header	28 KB
capstone.dll	2019/1/10 21:54	应用程序扩展	3,758 KB
capstone.h	2019/7/19 14:34	C/C++ Header	29 KB
capstone.lib	2019/1/10 21:54	Object File Library	6 KB
CapstoneDemo.cpp	2019/7/19 14:51	c_file	1 KB
CapstoneDemo.sln	2019/7/19 13:17	Visual Studio Sol...	2 KB
CapstoneDemo.vcxproj	2019/7/19 14:33	VC++ Project	8 KB
CapstoneDemo.vcxproj.filters	2019/7/19 14:33	VC++ Project Fil...	2 KB
CapstoneDemo.vcxproj.user	2019/7/19 13:17	Per-User Project...	1 KB
evm.h	2019/1/10 21:45	C/C++ Header	5 KB
m68k.h	2019/1/10 21:45	C/C++ Header	14 KB
m680x.h	2019/1/10 21:45	C/C++ Header	13 KB
mips.h	2019/1/10 21:45	C/C++ Header	17 KB
platform.h	2019/1/10 21:45	C/C++ Header	4 KB
ppc.h	2019/1/10 21:45	C/C++ Header	26 KB
sparc.h	2019/1/10 21:45	C/C++ Header	12 KB
systemz.h	2019/1/10 21:45	C/C++ Header	14 KB
tms320c64x.h	2019/1/10 21:45	C/C++ Header	9 KB
x86.h	2019/1/10 21:45	C/C++ Header	42 KB
xcore.h	2019/1/10 21:45	C/C++ Header	5 KB

In the VS solution, the header file adds the existing item capstone.h, add capstone to the resource file.lib, regenerate the solution



So let's test our own capstone engine now

Write the following code to the main file

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>
#INCLUDE <CINTTYPES>
#INCLUDE "CAPSTONE.H"
USING NAMESPACE STD;

#DEFINE CODE "\X55\X48\X8B\X05\XB8\X13\X00\X00"

INT MAIN(VOID)
{
	CSH HANDLE;
	CS_INSN* INSN;
	SIZE_T COUNT;

	IF (CS_OPEN(CS_ARCH_X86, CS_MODE_64, &HANDLE)) {
		PRINTF("ERROR: FAILED TO INITIALIZE ENGINE!\N");
		RETURN -1;
	}

	COUNT = CS_DISASM(HANDLE, (UNSIGNED CHAR*)CODE, SIZEOF(CODE) - 1, 0X1000, 0, &INSN);
	IF (COUNT) {
		SIZE_T J;

		FOR (J = 0; J < COUNT; J++) {
			PRINTF("0X%""IX"":\T%S\T\T%S\N", INSN[J].ADDRESS, INSN[J].MNEMONIC,
INSN[J].OP_STR);
		}

		CS_FREE(INSN, COUNT);
	}
	ELSE
		PRINTF("ERROR: FAILED TO DISASSEMBLE GIVEN CODE!\N");

	CS_CLOSE(&HANDLE);

	RETURN 0;
}
```

Screen of code(is hidden):

Result:

0x1 data type

csh

- Used to generate a handle to call the capstone API

```
size_t csh
```

Note: csh handle;

cs_arch

-Architecture selection

Code:

```
ENUM CS_ARCH {
        CS_ARCH_ARM = 0, ///< ARM ( Thumb, Thumb-2)
        CS_ARCH_ARM64,           ///< ARM-64, AArch64
        CS_ARCH_MIPS,            ///< Mips
  CS_ARCH_X86,          ///< X86  ( x86 & x86-64)
        CS_ARCH_PPC,             ///< PowerPC
        CS_ARCH_SPARC,           ///< Sparc
        CS_ARCH_SYSZ,            ///< SystemZ
        CS_ARCH_XCORE,           ///< XCore
        CS_ARCH_M68K,            ///< 68K
        CS_ARCH_TMS320C64X,      ///< TMS320C64x
        CS_ARCH_M680X,           ///< 680X
        CS_ARCH_EVM,             ///< Ethereum
        CS_ARCH_MAX,
        CS_ARCH_ALL = 0xFFFF, // All architecture - for cs_support()
} cs_arch;
```

Note: Fill in the enumeration content for the cs_arch parameter in the API, such as cs_open (cs_arch arch, cs_mode mode, csh *handle) in the API; fill in CS_ARCH_X86 for the first parameter to support the X86 architecture

cs_mode

- Mode selection

```
enum cs_mode {
        CS_MODE_LITTLE_ENDIAN = 0,          ///< little-endian 模式 (default 模式)
        CS_MODE_ARM = 0,                          ///< 32-bit ARM
        CS_MODE_16 = 1 << 1,                   ///< 16-bit 模式 (X86)
        CS_MODE_32 = 1 << 2,                   ///< 32-bit 模式 (X86)
        CS_MODE_64 = 1 << 3,                   ///< 64-bit 模式 (X86, PPC)
        CS_MODE_THUMB = 1 << 4,                ///< ARM's Thumb 模式, 包括 Thumb-2
        CS_MODE_MCLASS = 1 << 5,            ///< ARM's Cortex-M 系列
        CS_MODE_V8 = 1 << 6,                   ///< ARMv8 A32解码方式
        CS_MODE_MICRO = 1 << 4,             ///< MicroMips 模式 (MIPS)
        CS_MODE_MIPS3 = 1 << 5,             ///< Mips III ISA
        CS_MODE_MIPS32R6 = 1 << 6,           ///< Mips32r6 ISA
        CS_MODE_MIPS2 = 1 << 7,             ///< Mips II ISA
        CS_MODE_V9 = 1 << 4,               ///< SparcV9 模式 (Sparc)
        CS_MODE_QPX = 1 << 4,              ///< Quad Processing eXtensions 模式 (PPC)
        CS_MODE_SPE = 1 << 5,              ///< Signal Processing Engine 模式 (PPC)
        CS_MODE_BOOKE = 1 << 6,             ///< Book-E 模式 (PPC)
        CS_MODE_M68K_000 = 1 << 1,           ///< M68K 68000 模式
        CS_MODE_M68K_010 = 1 << 2,           ///< M68K 68010 模式
        CS_MODE_M68K_020 = 1 << 3,           ///< M68K 68020 模式
        CS_MODE_M68K_030 = 1 << 4,           ///< M68K 68030 模式
        CS_MODE_M68K_040 = 1 << 5,           ///< M68K 68040 模式
        CS_MODE_M68K_060 = 1 << 6,           ///< M68K 68060 模式
        CS_MODE_BIG_ENDIAN = 1 << 31,                 ///< big-endian 模式
        CS_MODE_MIPS32 = CS_MODE_32,                  ///< Mips32 ISA (Mips)
        CS_MODE_MIPS64 = CS_MODE_64,                  ///< Mips64 ISA (Mips)
        CS_MODE_M680X_6301 = 1 << 1,          ///< M680X Hitachi 6301,6303 模式
        CS_MODE_M680X_6309 = 1 << 2,          ///< M680X Hitachi 6309 模式
        CS_MODE_M680X_6800 = 1 << 3,          ///< M680X Motorola 6800,6802 模式
        CS_MODE_M680X_6801 = 1 << 4,          ///< M680X Motorola 6801,6803 模式
        CS_MODE_M680X_6805 = 1 << 5,          ///< M680X Motorola/Freescale 6805 模式
        CS_MODE_M680X_6808 = 1 << 6,          ///< M680X Motorola/Freescale/NXP 68HC08 模式
        CS_MODE_M680X_6809 = 1 << 7,          ///< M680X Motorola 6809 模式
        CS_MODE_M680X_6811 = 1 << 8,          ///< M680X Motorola/Freescale/NXP 68HC11 模式
        CS_MODE_M680X_CPU12 = 1 << 9,         ///< M680X Motorola/Freescale/NXP CPU12
                                                                        ///< 用于 M68HC12/HCS12
        CS_MODE_M680X_HCS08 = 1 << 10,         ///< M680X Freescale/NXP HCS08 模式
        CS_MODE_BPF_CLASSIC = 0,            ///< Classic BPF 模式 (默认)
        CS_MODE_BPF_EXTENDED = 1 << 0,                ///< Extended BPF 模式
        CS_MODE_RISCV32  = 1 << 0,           ///< RISCV RV32G
```

```
        CS_MODE_RISCV64  = 1 << 1,            ///< RISCV RV64G
        CS_MODE_RISCVC   = 1 << 2,            ///< RISCV 压缩指令模式
        CS_MODE_MOS65XX_6502 = 1 << 1,         ///< MOS65XXX MOS 6502
        CS_MODE_MOS65XX_65C02 = 1 << 2,        ///< MOS65XXX WDC 65c02
        CS_MODE_MOS65XX_W65C02 = 1 << 3,       ///< MOS65XXX WDC W65c02
        CS_MODE_MOS65XX_65816 = 1 << 4,        ///< MOS65XXX WDC 65816, 8-bit m/x
        CS_MODE_MOS65XX_65816_LONG_M = (1 << 5), ///< MOS65XXX WDC 65816, 16-bit m, 8-bit x
        CS_MODE_MOS65XX_65816_LONG_X = (1 << 6), ///< MOS65XXX WDC 65816, 8-bit m, 16-bit x
        CS_MODE_MOS65XX_65816_LONG_MX = CS_MODE_MOS65XX_65816_LONG_M | CS_MODE_MOS65XX_65816_LONG_X,
} CS_MODE;
```

Note: Fill in the enumeration content for the cs_mode parameter in the API, such as cs_open (cs_arch arch, cs_mode mode, csh *handle) in the API; fill in CS_MODE_64 for the second parameter to support X64 mode

cs_opt_mem

-Memory operation

```
STRUCT CS_OPT_MEM {
        CS_MALLOC_T MALLOC;
        CS_CALLOC_T CALLOC;
        CS_REALLOC_T REALLOC;
        CS_FREE_T FREE;
        CS_VSNPRINTF_T VSNPRINTF;
} CS_OPT_MEM;
```

Note: You can use the user-defined malloc/calloc/realloc/free/vsnprintf() function. By default, the system comes with malloc (), calloc(), realloc(), free() & vsnprintf().

cs_opt_mnem

- Custom mnemonics

```
STRUCT CS_OPT_MNEM {
        /// Need a custom instruction ID
        unsigned int ID;
        /// Custom mnemonics
        const char *mnemonic;
} CS_OPT_MNEM;
```

cs_opt_type

-Decompiled runtime options

```
ENUM CS_OPT_TYPE {
        CS_OPT_INVALID = 0,   //No special requirements
        CS_OPT_SYNTAX,       //Assembly output syntax
        CS_OPT_DETAIL,      //Break down the instruction structure into multiple details
        CS_OPT_MODE,       //Change engine mode at runtime
        CS_OPT_MEM,       //User-defined dynamic memory-related functions
        CS_OPT_SKIPDATA, //Skip data when disassembling.Then the engine will be in SKIPDATA mode
        CS_OPT_SKIPDATA_SETUP,  //Set user-defined functions for the SKIPDATA option
        CS_OPT_MNEMONIC,       //Custom instruction mnemonics
```

```
        CS_OPT_UNSIGNED,       //Print immediate operands in unsigned form
} cs_opt_type;
```

Not called in the currently open API

cs_ac_type

-Common instruction operand access types, consistent in all architectures, access types can be combined, for example: CS_AC_READ | CS_AC_WRITE

Code:

```
enum cs_ac_type {
        CS_AC_INVALID = 0,       //Uninitialized/invalid access type
        CS_AC_READ    = 1 << 0,  //The operand is read from memory or register
        CS_AC_WRITE   = 1 << 1,  //The operand is written from memory or register
} cs_ac_type;
```

Not called in the currently open API

cs_group_type

- Common instruction set, consistent across all architectures

Code:

```
cs_group_type {
        CS_GRP_INVALID = 0,  //Uninitialized/invalid instruction set
        CS_GRP_JUMP,    //All jump instructions (conditional jump + direct jump + indirect jump)
        CS_GRP_CALL,    //All calling instructions
        CS_GRP_RET,     //All return instructions
        CS_GRP_INT,     //All interrupt instructions (INT+SYSCALL)
        CS_GRP_IRET,    //All interrupt return instructions
        CS_GRP_PRIVILEGE,    //All privileged instructions
        CS_GRP_BRANCH_RELATIVE, //All relevant branch instructions
} cs_group_type;
```

Not called in the currently open API

cs_opt_skipdata

-User-defined settings SKIPDATA options

Code:

```
struct cs_opt_skipdata {
        /// CAPSTONE BELIEVES THAT THE DATA TO BE SKIPPED IS A SPECIAL "INSTRUCTION”
        /// The user can specify the “mnemonic” string of the instruction here
        /// By default (@mnemonic is NULL), CAPSTONE USES ".BYTE"
        const char *mnemonic;

        /// User-defined callback function, called when CAPSTONE HITS DATA
        /// IF THE VALUE RETURNED BY THIS CALLBACK IS A POSITIVE number (>0), CAPSTONE WILL
SKIP THIS NUMBER OF BYTES AND CONTINUE.If the callback returns 0, CAPSTONE will stop disassembling
and immediately return from CS_DISASM()
        /// Note: If THIS CALLBACK POINTER IS EMPTY, CAPSTONE WILL SKIP SOME BYTES
ACCORDING TO THE ARCHITECTURE, AS SHOWN BELOW:
        /// Arm:     2 bytes (Thumb mode) or 4 bytes.
        /// Arm64:   4 bytes.
        /// Mips:    4 bytes.
        /// M680x:   1 byte.
        /// PowerPC: 4 bytes.
        /// Sparc:   4 bytes.
        /// SystemZ: 2 bytes.
        /// X86:     1 bytes.
        /// XCore:   2 bytes.
        /// EVM:     1 bytes.
        /// RISCV:   4 bytes.
        /// WASM:    1 bytes.
        /// MOS65XX: 1 bytes.
        /// BPF:     8 bytes.
        cs_skipdata_cb_t callback;     // The default value is NULL

        /// User-defined data will be passed to the @callback function pointer
        void *user_data;
} cs_opt_skipdata;
```

Not called in the currently open API

cs_detail

-Note: All information in cs_detail is only available when CS_OPT_DETAIL = CS_OPT_ON

In arch/ARCH/ARCHDisassembler.The ARCH_getInstruction of c is initialized to memset (., 0, offsetof(cs_detail, ARCH)+sizeof(cs_ARCH))

If cs_detail changes, especially if a field is added after union, then update arch/arch/archdisassembly accordingly.c

Code:

```
struct cs_detail {
        uint16_t regs_read[16]; ///< This parameter reads the list of implicit registers
        uint8_t regs_read_count; ///< This parameter reads the implicit register count

        uint16_t regs_write[20]; ///< This parameter modifies the list of implicit registers
        uint8_t regs_write_count; ///< This parameter modifies the implicit register count

        uint8_t groups[8]; ///< List of instruction groups to which this instruction belongs
        uint8_t groups_count; ///< The number of groups to which this instruction belongs

        /// Architecture-specific information
        union {
                cs_x86 x86;     ///< X86, 16-bit, 32-bit & 64-bit
                cs_arm64 arm64; ///< ARM64(aka AArch64)
                cs_arm arm;     ///< ARM(Thumb/Thumb2)
                cs_m68k m68k;   ///< M68K
                cs_mips mips;   ///< MIPS
                cs_ppc ppc;             ///< PowerPC
                cs_sparc sparc; ///< Sparc
                cs_sysz sysz;   ///< SystemZ
                cs_xcore xcore; ///< XCore
                cs_tms320c64x tms320c64x;  ///< TMS320C64x
                cs_m680x m680x; ///< M680X
                cs_evm evm;             ///< Ethereum
                cs_mos65xx mos65xx;///< MOS65XX(MOS6502)
                cs_wasm wasm;       ///< Web Assembly
                cs_bpf bpf;         ///< Berkeley Packet Filter (eBPF)
                cs_riscv riscv; ///< RISCV
        };
} cs_detail;
```

cs_insn

- Instruction details

Code:

```
struct cs_insn {
        /// Instruction ID (basically a numeric ID used for instruction mnemonics)
        /// The instruction ID in'[ARCH]_INSN'enum should be found in the header file of the corresponding architecture, such as arm.The'ARM_INSN' in h represents arm, x86.The 'X86_INSN' in h represents x86, etc....
        /// This information can be used even when cs_opt_detail = cs_opt_off
        /// Note: In skipdata mode, the "DATA" command for this ID field is 0
        unsigned int id;

        /// Instruction address (EIP)
        /// This information can be used even when cs_opt_detail = cs_opt_off
        uint64_t address;

        /// Instruction length
        /// This information can be used even when cs_opt_detail = cs_opt_off
        uint16_t size;

        /// The machine code of this instruction, the number of bytes is represented by the @size above
        /// This information can be used even when cs_opt_detail = cs_opt_off
        uint8_t bytes[24];

        /// ASCII text mnemonic for instructions
        /// This information can be used even when cs_opt_detail = cs_opt_off
        char mnemonic[cs_mnemonic_size];

        /// ASCII text for instruction operands
        /// This information can be used even when cs_opt_detail = cs_opt_off
        char op_str[160];

        /// cs_detail pointer
```

```
        /// Note: The DETAIL pointer is only valid if the following two requirements are met at the same time:
        /// (1) CS_OP_DETAIL = CS_OPT_ON
        /// (2) The engine is not in SKIPDATA mode (the CS_OP_SKIPDATA option is set to CS_OPT_ON)
        /// Note 2: When in SKIPDATA mode or DETAIL mode is turned off, even if this pointer is not NULL, its content is still irrelevant.
        cs_detail *detail;
} cs_insn;
```

cs_err

-The return value of cs_errno() for various types of errors encountered by the Capstone API

Code:

```
typedef enum cs_err {
        CS_ERR_OK = 0,    ///< No error
        CS_ERR_MEM,       ///< Insufficient memory: cs_open(), cs_disasm(), cs_disasm_iter()
        CS_ERR_ARCH,      ///< Unsupported architecture: cs_open()
        CS_ERR_HANDLE,    ///< The handle is not available: cs_op_data(), cs_op_index()
        CS_ERR_CSH, //csh parameters are not available: cs_close(), cs_errno(), cs_option()
        CS_ERR_MODE,      ///< Invalid or unsupported mode: cs_open()
        CS_ERR_OPTION,    ///< Invalid or unsupported options: cs_option()
        CS_ERR_DETAIL,    ///< The information is not available because the DETAIL option is turned off
        CS_ERR_MEMSETUP,  //Dynamic memory management is not initialized (see CS_OPT_MEM)
        CS_ERR_VERSION,   ///< UNSUPPORTED version (bindings)
        CS_ERR_DIET,      ///< Access irrelevant data in the "diet" engine
        CS_ERR_SKIPDATA,  ///< Access data unrelated to the "data" instruction in SKIPDATA mode
        CS_ERR_X86_ATT,   ///< X86 AT&T syntax is not supported (exit at compile time)
        CS_ERR_X86_INTEL, ///< X86 INTEL syntax is not supported (exit at compile time)
        CS_ERR_X86_MASM,  ///< X86 INTEL syntax is not supported (exit at compile time)
} cs_err;
```

0x2 API

cs_version

```
unsigned int CAPSTONE_API cs_version(int *major, int *minor);
```

Used to output the capstone version number:

Major: API main version

Minor: API minor version

Return: Returns the hexadecimal of the primary and secondary versions, such as version 4.0 returns 0x0400

This version is defined in cs.In c, it cannot be changed after compilation, and custom versions are not accepted

```
unsigned int CAPSTONE_API cs_version(int *major, int *minor)
{
    if (major != NULL && minor != NULL) {
        *major = CS_API_MAJOR;
        *minor = CS_API_MINOR;
    }

    return (CS_API_MAJOR << 8) + CS_API_MINOR;
}
```

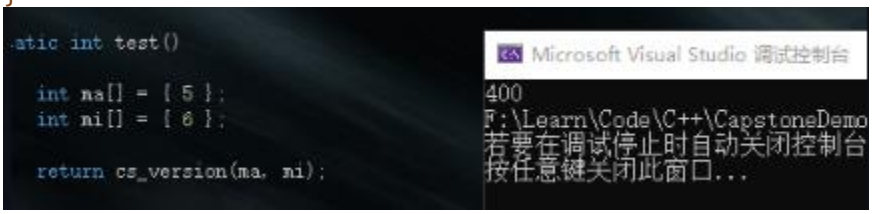
```
// Capstone API version
#define CS_API_MAJOR 4
#define CS_API_MINOR 0

// Version for bleeding edge code of the Github's "next" branch.
// Use this if you want the absolutely latest development code.
// This version number will be bumped up whenever we have a new major change.
#define CS_NEXT_VERSION 5

// Capstone package version
#define CS_VERSION_MAJOR CS_API_MAJOR
#define CS_VERSION_MINOR CS_API_MINOR
#define CS_VERSION_EXTRA 1
```

Code:

```
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>

#INCLUDE "PLATFORM.H"
#INCLUDE "CAPSTONE.H"

STATIC INT TEST()
{
        RETURN CS_VERSION(NULL, NULL);
}

INT MAIN()
{
        INT VERSION = TEST();
        PRINTF("%X", VERSION);
        RETURN 0;
}
```

Example 2, forcibly modify the version:

```
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>

#INCLUDE "PLATFORM.H"
#INCLUDE "CAPSTONE.H"

STATIC INT TEST()
{
        INT MA[] = { 5 };
        INT MI[] = { 6 };

        RETURN CS_VERSION(MA, MI);
}

INT MAIN()
{
        INT VERSION = TEST();
        PRINTF("%X", VERSION);
        RETURN 0;
}
```



Visible and cannot be changed

cs_support

```
bool CAPSTONE_API cs_support(int query);
```

Used to check whether the capstone library supports the architecture of parameter input or is in a certain compilation option

Code:

```
BOOL CAPSTONE_API CS_SUPPORT(INT QUERY)

{
        IF (QUERY == CS_ARCH_ALL)
                RETURN ALL_ARCH == ((1 << CS_ARCH_ARM) | (1 << CS_ARCH_ARM64) |
                                    (1 << CS_ARCH_MIPS) | (1 << CS_ARCH_X86) |
                                    (1 << CS_ARCH_PPC) | (1 << CS_ARCH_SPARC) |
                                    (1 << CS_ARCH_SYSZ) | (1 << CS_ARCH_XCORE) |
                                    (1 << CS_ARCH_M68K) | (1 << CS_ARCH_TMS320C64X) |
                                    (1 << CS_ARCH_M680X) | (1 << CS_ARCH_EVM));

        IF ((UNSIGNED INT)QUERY < CS_ARCH_MAX)
                RETURN ALL_ARCH & (1 << QUERY);

        IF (QUERY == CS_SUPPORT_DIET) {
#IFDEF CAPSTONE_DIET
                RETURN TRUE;
#ELSE
                RETURN FALSE;
#ENDIF
        }

        IF (QUERY == CS_SUPPORT_X86_REDUCE) {
#IF DEFINED(CAPSTONE_HAS_X86) && DEFINED(CAPSTONE_X86_REDUCE)
                RETURN TRUE;
#ELSE
                RETURN FALSE;
#ENDIF
        }

        // UNSUPPORTED QUERY
        RETURN FALSE;
}
```

Example 1 (CS_ARCH_ALL, check whether all architectures are supported)

```
#include <stdio.h>
#include <stdlib.h>

#include "platform.h"
#include "capstone.h"

static int test()
{
    return cs_support(CS_ARCH_ALL);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
1
F:\Le
若要在
按任意
```

Example 2 (CS_ARCH_*, check whether the specified architecture is supported)

```
#include <stdio.h>
#include <stdlib.h>

#include "platform.h"
#include "capstone.h"

static int test()
{
    return cs_support(CS_ARCH_X86);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
1
F:\Le
若要在
按任意
```

Example 3 (check if it is in DIET compilation mode) ：

```
#include <stdio.h>
#include <stdlib.h>

#include "platform.h"
#include "capstone.h"

static int test()
{
    return cs_support(CS_SUPPORT_DIET);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
0
F:\
若要
按任
```

Example 4 (check if it is in X86_REDUCE compilation mode):

```
#include <stdio.h>
#include <stdlib.h>

#include "platform.h"
#include "capstone.h"

static int test()
{
    return cs_support(CS_SUPPORT_X86_REDUCE);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
0
F:\Le
若要在
按任意
```

cs_malloc_t

```
void* (CAPSTONE_API *cs_malloc_t)(size_t size);
```

Dynamic memory allocation of cs, used for

```
STRUCT CS_OPT_MEM {
        CS_MALLOC_T MALLOC;
        CS_CALLOC_T CALLOC;
        CS_REALLOC_T REALLOC;
        CS_FREE_T FREE;
        CS_VSNPRINTF_T VSNPRINTF;
} CS_OPT_MEM;
```

In user mode, cs_mem_malloc uses system malloc by default

In Windows driver mode, cs_malloc_t cs_mem_malloc = cs_winkernel_malloc;

cs_winkernel_malloc is defined in \capstone-4.0.1\windows\winkernel_mm.c,

Code:

```
VOID * CAPSTONE_API CS_WINKERNEL_MALLOC(SIZE_T SIZE)

{
        // 长度不能分配为0
        NT_ASSERT(SIZE);

        // FP; NONPAGEDPOOL用于支持 WINDOWS 7
#PRAGMA PREFAST(SUPPRESS : 30030)              // 分配可执行的POOL_TYPE内存
        SIZE_T NUMBER_OF_BYTES = 0;
        CS_WINKERNEL_MEMBLOCK *BLOCK = NULL;
        // 特定的值能造成溢出
        // 如果VALUE中的和超出或低于类型容量，函数将返回NULL。
        IF (!NT_SUCCESS(RTLSIZETADD(SIZE, SIZEOF(CS_WINKERNEL_MEMBLOCK),
&NUMBER_OF_BYTES))) {
                RETURN NULL;
        }
        BLOCK = (CS_WINKERNEL_MEMBLOCK *)EXALLOCATEPOOLWITHTAG(
                        NONPAGEDPOOL, NUMBER_OF_BYTES, CS_WINKERNEL_POOL_TAG);
        IF (!BLOCK) {
                RETURN NULL;
```

```
        }
        BLOCK->SIZE = SIZE;

        RETURN BLOCK->DATA;
}
```

In OSX kernel mode, cs_malloc_t cs_mem_malloc = kern_os_malloc;, will not be discussed here for the time being.

cs_calloc_t

```
void* (CAPSTONE_API *cs_calloc_t)(size_t nmemb, size_t size);
```

cs applies for memory and initializes

Used for struct cs_opt_mem, defined in cs.c

User mode: cs_calloc_t cs_mem_calloc = calloc;, use system calloc

Windows driver mode: cs_calloc_t cs_mem_calloc = cs_winkernel_calloc;

Code:

```
VOID * CAPSTONE_API CS_WINKERNEL_CALLOC(SIZE_T N, SIZE_T SIZE)
{
        SIZE_T TOTAL = N * SIZE;

        VOID *NEW_PTR = CS_WINKERNEL_MALLOC(TOTAL);
        IF (!NEW_PTR) {
                RETURN NULL;
        }

        RETURN RTLFILLMEMORY(NEW_PTR, TOTAL, 0);
}
```

OSX kernel mode: cs_calloc_t cs_mem_calloc = cs_kern_os_calloc; directly call kern_os_malloc

cs_realloc_t

```
void* (CAPSTONE_API *cs_realloc_t)(void *ptr, size_t size);
```

cs reallocates memory

Used for struct cs_opt_mem, defined in cs.c

User mode: cs_realloc_t cs_mem_realloc= realloc;, call the system realloc

Windows driver mode: cs_realloc_t cs_mem_realloc = cs_winkernel_realloc;

Code:

```
VOID * CAPSTONE_API CS_WINKERNEL_REALLOC(VOID *PTR, SIZE_T SIZE)
{
        VOID *NEW_PTR = NULL;
        SIZE_T CURRENT_SIZE = 0;
        SIZE_T SMALLER_SIZE = 0;

        IF (!PTR) {
                RETURN CS_WINKERNEL_MALLOC(SIZE);
        }

        NEW_PTR = CS_WINKERNEL_MALLOC(SIZE);
        IF (!NEW_PTR) {
                RETURN NULL;
        }

        CURRENT_SIZE = CONTAINING_RECORD(PTR, CS_WINKERNEL_MEMBLOCK, DATA)->SIZE;
        SMALLER_SIZE = (CURRENT_SIZE < SIZE) ? CURRENT_SIZE : SIZE;
        RTLCOPYMEMORY(NEW_PTR, PTR, SMALLER_SIZE);
        CS_WINKERNEL_FREE(PTR);

        RETURN NEW_PTR;
}
```

OSX kernel mode: cs_realloc_t cs_mem_realloc = kern_os_realloc;

cs_free_t

```
typedef void (CAPSTONE_API *cs_free_t)(void *ptr);
```

cs frees up memory

Used for struct cs_opt_mem, defined in cs.c

User mode: cs_free_t cs_mem_free =free;, call the system free

Windows driver mode: cs_free_t cs_mem_free = cs_winkernel_free;

```
VOID CAPSTONE_API CS_WINKERNEL_FREE(VOID *PTR)
{
        IF (PTR) {
                EXFREEPOOLWITHTAG(CONTAINING_RECORD(PTR, CS_WINKERNEL_MEMBLOCK, DATA),
CS_WINKERNEL_POOL_TAG);
        }
}
```

OSX kernel mode: cs_free_t cs_mem_free = kern_os_free;

cs_vsnprintf_t

```
int (CAPSTONE_API *cs_vsnprintf_t)(char *str, size_t size, const char *format,
va_list ap);
```

Output to the string str by size

if the system is wince, the _vsnprintf function will be used

vsnprintf () and _vsnprintf() are both available for drivers, but they have some differences

Use vsnprintf() when you need to return a value and set a null terminator

Windows driver mode: cs_vsnprintf_t cs_vsnprintf = cs_winkernel_vsnprintf;

```
INT CAPSTONE_API CS_WINKERNEL_VSNPRINTF(CHAR *BUFFER, SIZE_T COUNT, CONST CHAR *FORMAT, VA_LIST
ARGPTR)
{
        INT RESULT = _VSNPRINTF(BUFFER, COUNT, FORMAT, ARGPTR);

            // _VSNPRINTF()RETURNS -1 WHEN THE STRING IS TRUNCATED, AND "COUNT" WHEN THE ENTIRE
STRING IS STORED BUT THERE IS NO "¥0" AT THE END OF "BUFFER".IN BOTH CASES, YOU NEED TO MANUALLY ADD A NULL
TERMINATOR.
        IF (RESULT == -1 || (SIZE_T)RESULT == COUNT) {
                BUFFER[COUNT - 1] = '\0';
        }

        IF (RESULT == -1) {
                // WHEN RETURNING -1, THE FUNCTION MUST OBTAIN AND RETURN SOME CHARACTERS THAT
WERE ORIGINALLY TO BE WRITTEN.THEREFORE, BY RETRY USING THE TEMP BUFFER FOR THE SAME
CONVERSION, THIS BUFFER MAY BE LARGE ENOUGH TO COMPLETE THE FORMATTING AND GET A LOT OF
CHARACTERS THAT SHOULD HAVE BEEN WRITTEN.
                CHAR* TMP = CS_WINKERNEL_MALLOC(0X1000);
                IF (!TMP) {
                        RETURN RESULT;
                }

                RESULT = _VSNPRINTF(TMP, 0X1000, FORMAT, ARGPTR);
                NT_ASSERT(RESULT != -1);
                CS_WINKERNEL_FREE(TMP);
        }

        RETURN RESULT;
}
```

OSX kernel mode: cs_vsnprintf_t cs_vsnprintf= vsnprintf;, use the default vsnprintf

cs_skipdata_cb_t

```
size_t (CAPSTONE_API *cs_skipdata_cb_t)(const uint8_t *code, size_t code_size, size_t
offset, void *user_data);
```

User-defined callback function for the SKIPDATA option.

Code: The input buffer that contains the code to be decomposed.Same as the buffer passed to cs_disasm().

code_size: The size of the code buffer above (in bytes).

offset: The position of the current check byte in the input buffer code mentioned above.

user_data: User data is passed to cs_option() through the @user_data field in the cs_opt_skipdata structure.

Return: Returns the number of bytes to be skipped, or 0 means that disassembly is stopped immediately.

cs_skipdata_cb_t is called in struct cs_opt_skipdata

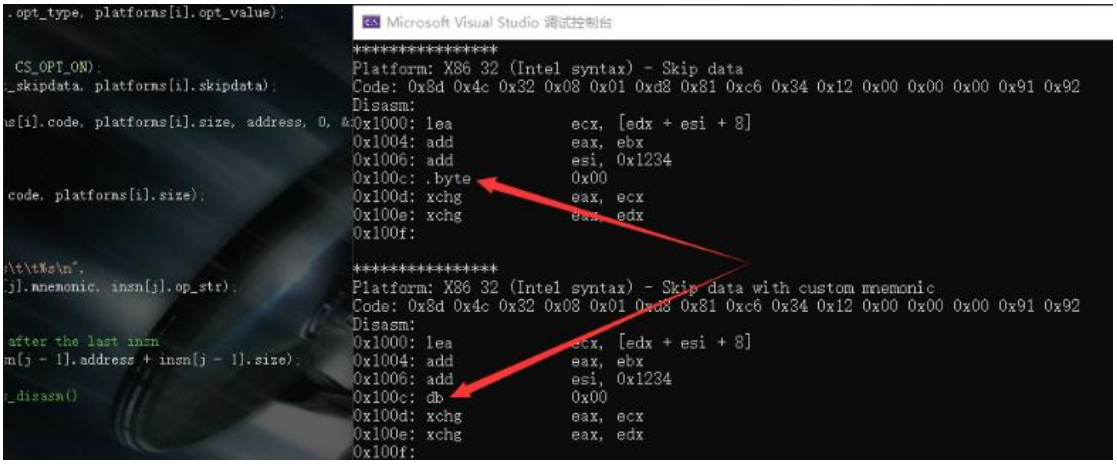
Code:

```
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>

#INCLUDE "PLATFORM.H"
#INCLUDE "CAPSTONE.H"

STRUCT PLATFORM {
          CS_ARCH ARCH;
          CS_MODE MODE;
          UNSIGNED CHAR* CODE;
          SIZE_T SIZE;
          CONST CHAR* COMMENT;
          CS_OPT_TYPE OPT_TYPE;
          CS_OPT_VALUE OPT_VALUE;
          CS_OPT_TYPE OPT_SKIPDATA;
          SIZE_T SKIPDATA;
};

STATIC VOID PRINT_STRING_HEX(UNSIGNED CHAR* STR, SIZE_T LEN)  //OUTPUT MACHINE CODE
{
          UNSIGNED CHAR* C;

          PRINTF("CODE: ");
          FOR (C = STR; C < STR + LEN; C++) {
                    PRINTF("0X%02X ", *C & 0XFF);
          }
          PRINTF("\N");
}

STATIC VOID TEST()
{

#DEFINE X86_CODE32 "\X8D\X4C\X32\X08\X01\XD8\X81\XC6\X34\X12\X00\X00\X00\X91\X92"  //
//MACHINE CODE FOR TESTING
```

```
#define RANDOM_CODE
"\xed\x00\x00\x00\x00\x1a\x5a\x0f\x1f\xff\xc2\x09\x80\x00\x00\x00\x07\xf7\xeb\x2a\xff\xff\x7f
\x57\xe3\x01\xff\xff\x7f\x57\xeb\x00\xf0\x00\x00\x24\xb2\x4f\x00\x78"

        cs_opt_skipdata skipdata = {
                // Change the default "DATA" descriptor from ".Rename "BYTE" To "DB" "DB",
        };

        struct platform platforms[2] = {        //Create an array in two ways: default descriptors and
custom descriptors
                {
                        CS_ARCH_X86,
                        CS_MODE_32,
                        (unsigned char*)X86_CODE32,
                        sizeof(X86_CODE32) - 1,
                        "X86 32 (Intel syntax) - Skip data",
                },
                {
                        CS_ARCH_X86,
                        CS_MODE_32,
                        (unsigned char*)X86_CODE32,
                        sizeof(X86_CODE32) - 1,
                        "X86 32 (Intel syntax) - Skip data with custom mnemonic",
                        CS_OPT_INVALID,
                        CS_OPT_OFF,
                        CS_OPT_SKIPDATA_SETUP,
                        (size_t)& skipdata,
                },

        };

        csh handle;    // Create a CAPSTONE handle
        uint64_t address = 0x1000;  // Set the starting address
        cs_insn* insn;  // Specific information structure
        cs_err err;  //Error enumeration
        int i;
        size_t count;  //Number of successfully disassembled lines

        for (i = 0; i < sizeof(platforms) / sizeof(platforms[0]); i++) {
                printf("****************\n");
                printf("Platform: %s\n", platforms[i].comment);
                err = cs_open(platforms[i].arch, platforms[i].mode, &handle);  //Error checking
                if (err) {
                        printf("Failed on cs_open() with error returned: %u\n", err);
                        abort();
                }

                if (platforms[i].opt_type)
                        cs_option(handle, platforms[i].opt_type, platforms[i].opt_value);
```

```
                    // Turn on SKIPDATA mode
                    cs_option(handle, CS_OPT_SKIPDATA, CS_OPT_ON);
                    cs_option(handle, platforms[i].opt_skipdata, platforms[i].skipdata);

                    count = cs_disasm(handle, platforms[i].code, platforms[i].size, address, 0,
&insn);
                    if (count) {
                              size_t j;

                              print_string_hex(platforms[i].code, platforms[i].size);
                              printf("Disasm:\n");

                              for (j = 0; j < count; j++) {  // Output assembly
                                        printf("0x%" PRIx64 ":\t%s\t\t%s\n",
                                                  insn[j].address, insn[j].mnemonic,
insn[j].op_str);
                              }

                              // Print the offset after the last line of code
                              printf("0x%" PRIx64 ":\n", insn[j - 1].address + insn[j - 1].size);

                              //Free up the memory requested by cs_disasm()
                              cs_free(insn, count);
                    }
                    else {
                              printf("****************\n");
                              printf("Platform: %s\n", platforms[i].comment);
                              print_string_hex(platforms[i].code, platforms[i].size);
                              printf("ERROR: Failed to disasm given code!\n");
                              abort();
                    }

                    printf("\n");

                    cs_close(&handle);
          }
}

int main()
{
          test();

          return 0;
}
```

The running result is as follows, the default.The byte data type is changed to the db descriptor



cs_open

```
cs_err CAPSTONE_API cs_open(cs_arch arch, cs_mode mode, csh *handle);
```

Initialize the cs handle

arch: Architecture type (CS_ARCH_*)

Mode: hardware mode. CS_MODE_* can be found in the cs_mode data type

handle: Points to the handle, updated when returned

Return: CS_ERR_OK is returned after successful creation, otherwise the corresponding error message in the cs_err enumeration is returned

Code:

```
CS_ERR CAPSTONE_API CS_OPEN(CS_ARCH ARCH, CS_MODE MODE, CSH *HANDLE)
{
        CS_ERR ERR;
        STRUCT CS_STRUCT *UD;
        IF (!CS_MEM_MALLOC || !CS_MEM_CALLOC || !CS_MEM_REALLOC || !CS_MEM_FREE || !CS_VSNPRINTF)
// ERROR: BEFORE USING CS_OPEN(), YOU MUST USE CS_OPTION(CS_OPT_MEM) TO INITIALIZE DYNAMIC
//MEMORY MANAGEMENT
                RETURN CS_ERR_MEMSETUP;

        IF (ARCH < CS_ARCH_MAX && CS_ARCH_INIT[ARCH]) {
                // VERIFY WHETHER THE ARCHITECTURE IS USED, METHOD: THE ARCHITECTURE IS IN THE
//ENUMERATION AND CAN BE INITIALIZED
                IF (MODE & CS_ARCH_DISALLOWED_MODE_MASK[ARCH]) {
                        *HANDLE = 0;
                        RETURN CS_ERR_MODE;
                }

                UD = CS_MEM_CALLOC(1, SIZEOF(*UD));
                IF (!UD) {
                        // INSUFFICIENT MEMORY
                        RETURN CS_ERR_MEM;
                }

                UD->ERRNUM = CS_ERR_OK;
                UD->ARCH = ARCH;
                UD->MODE = MODE;
                // BY DEFAULT, THE COMMAND DOES NOT TURN ON THE DETAIL MODE
```

```
                UD->DETAIL = CS_OPT_OFF;

                // DEFAULT SKIPDATA SETTING
                UD->SKIPDATA_SETUP.MNEMONIC = SKIPDATA_MNEM;

                ERR = CS_ARCH_INIT[UD->ARCH](UD);
                IF (ERR) {
                        CS_MEM_FREE(UD);
                        *HANDLE = 0;
                        RETURN ERR;
                }

                *HANDLE = (UINTPTR_T)UD;

                RETURN CS_ERR_OK;
        } ELSE {
                *HANDLE = 0;
                RETURN CS_ERR_ARCH;
        }
}
```

Among them, the cs_struct structure contains more detailed settings, as follows:

Code:

```
STRUCT CS_STRUCT {
        CS_ARCH ARCH;
        CS_MODE MODE;
        PRINTER_T PRINTER;  // PRINT ASM
        VOID *PRINTER_INFO; // PRINT INFORMATION
        DISASM_T DISASM;    // DECOMPILE
        VOID *GETINSN_INFO; // PRINT AUXILIARY INFORMATION
        GETNAME_T REG_NAME;
        GETNAME_T INSN_NAME;
        GETNAME_T GROUP_NAME;
        GETID_T INSN_ID;
        POSTPRINTER_T POST_PRINTER;
        CS_ERR ERRNUM;
        ARM_ITSTATUS ITBLOCK;       // ARM SPECIAL OPTIONS
        CS_OPT_VALUE DETAIL, IMM_UNSIGNED;
        INT SYNTAX;// BASIC ASM SYNTAX PRINTING FOR ARM, MIPS & PPC AND OTHER ARCHITECTURES
        BOOL DOING_MEM;     // PROCESS MEMORY OPERANDS IN INSTPRINTER CODE
        UNSIGNED SHORT *INSN_CACHE;   // FOR MAPPING.C ESTABLISH A CACHE INDEX
        GETREGISTERNAME_T GET_REGNAME;
        BOOL SKIPDATA;      // IF YOU WANT TO SKIP DATA WHEN DECOMPILING, SET THIS ITEM TO TRUE
```

```
        uint8_t skipdata_size;          // The number of bytes to skip
        cs_opt_skipdata skipdata_setup;         // Customize skipdata settings
        const uint8_t *regsize_map;     // Map register size (currently only supports x86)
        GetRegisterAccess_t reg_access;
        struct insn_mnem *mnem_list;    // List of links to custom instruction mnemonics
};
```

Example (create a cs handle of type x86_64)

```
cs_open(CS_ARCH_X86, CS_MODE_64, &handle)
```

cs_close

```
cs_err CAPSTONE_API cs_close(csh *handle);
```

Release handle

handle: Points to a handle opened by cs_open()

Return: The release successfully returns CS_ERR_OK, otherwise the error message of cs_err_ok is returned

The essence of releasing the handle is to set the handle value to 0.

Code:

```
cs_err CAPSTONE_API cs_close(csh *handle)
{
        struct cs_struct *ud;
        struct insn_mnem *next, *tmp;

        if (*handle == 0)
                // Handle unavailable
                return CS_ERR_CSH;

        ud = (struct cs_struct *)(*handle);

        if (ud->printer_info)
                cs_mem_free(ud->printer_info);

        // 释放自定义助记符的链接list
        tmp = ud->mnem_list;
        while(tmp) {
                next = tmp->next;
                cs_mem_free(tmp);
                tmp = next;
        }
```

```
        CS_MEM_FREE(UD->INSN_CACHE);

        MEMSET(UD, 0, SIZEOF(*UD));
        CS_MEM_FREE(UD);

        //THE HANDLE VALUE IS SET TO 0 TO ENSURE THAT THIS HANDLE CANNOT BE USED AFTER CS_CLOSE()
//IS RELEASED
        *HANDLE = 0;

        RETURN CS_ERR_OK;
}
```

Example

```
cs_close(&handle);
```

cs_option

```
cs_err CAPSTONE_API cs_option(csh handle, cs_opt_type type, size_t value);
```

Runtime options for decompiling the engine

Handle: cs_open() Open handle

type: The type of setting option

Value: The option value corresponding to type

Return: The setting successfully returns CS_ERR_OK, otherwise the error message of cs_err_ok is returned

Note: In the case of CS_OPT_MEM, handle can be any value, so cs_option (handle, CS_OPT_MEM, value) must be called before cs_open()

Code:

```
CS_ERR CAPSTONE_API CS_OPTION(CSH UD, CS_OPT_TYPE TYPE, SIZE_T VALUE)
{
        STRUCT CS_STRUCT *HANDLE;
        CS_OPT_MNEM *OPT;

        // SUPPORT SUPPORT IN FRONT OF ALL APIS (EVEN CS_OPEN())
        IF (TYPE == CS_OPT_MEM) {
                CS_OPT_MEM *MEM = (CS_OPT_MEM *)VALUE;

                CS_MEM_MALLOC = MEM->MALLOC;
                CS_MEM_CALLOC = MEM->CALLOC;
                CS_MEM_REALLOC = MEM->REALLOC;
                CS_MEM_FREE = MEM->FREE;
                CS_VSNPRINTF = MEM->VSNPRINTF;

                RETURN CS_ERR_OK;
        }

        HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;
```

```
	IF (!HANDLE)
		RETURN CS_ERR_CSH;

	SWITCH(TYPE) {
		DEFAULT:
			BREAK;

		CASE CS_OPT_UNSIGNED:
			HANDLE->IMM_UNSIGNED = (CS_OPT_VALUE)VALUE;
			RETURN CS_ERR_OK;

		CASE CS_OPT_DETAIL:
			HANDLE->DETAIL = (CS_OPT_VALUE)VALUE;
			RETURN CS_ERR_OK;

		CASE CS_OPT_SKIPDATA:
			HANDLE->SKIPDATA = (VALUE == CS_OPT_ON);
			IF (HANDLE->SKIPDATA) {
				IF (HANDLE->SKIPDATA_SIZE == 0) {
					HANDLE->SKIPDATA_SIZE = SKIPDATA_SIZE(HANDLE);
				}
			}
			RETURN CS_ERR_OK;

		CASE CS_OPT_SKIPDATA_SETUP:
			IF (VALUE)
				HANDLE->SKIPDATA_SETUP = *((CS_OPT_SKIPDATA *)VALUE);
			RETURN CS_ERR_OK;

		CASE CS_OPT_MNEMONIC:
			OPT = (CS_OPT_MNEM *)VALUE;
			IF (OPT->ID) {
				IF (OPT->MNEMONIC) {
					STRUCT INSN_MNEM *TMP;

			// ADD A NEW INSTRUCTION OR REPLACE AN EXISTING INSTRUCTION
					// VIEW THE CURRENT INSN RELEASE IN THE LIST
					TMP = HANDLE->MNEM_LIST;
					WHILE(TMP) {
						IF (TMP->INSN.ID == OPT->ID) {
					// FIND THE INSTRUCTION, REPLACE THE MNEMONIC
							(VOID)STRNCPY(TMP-
>INSN.MNEMONIC, OPT->MNEMONIC, SIZEOF(TMP->INSN.MNEMONIC) - 1);
							TMP->INSN.MNEMONIC[SIZEOF(TMP-
>INSN.MNEMONIC) - 1] = '\0';
							BREAK;
						}
						TMP = TMP->NEXT;
					}

					// 2. IF NOT, ADD THIS INSTRUCTION
					IF (!TMP) {
						TMP = CS_MEM_MALLOC(SIZEOF(*TMP));
```

```
                                                        TMP->INSN.ID = OPT->ID;
                                                        (VOID)STRNCPY(TMP->INSN.MNEMONIC, OPT-
>MNEMONIC, SIZEOF(TMP->INSN.MNEMONIC) - 1);
                                                        TMP->INSN.MNEMONIC[SIZEOF(TMP-
>INSN.MNEMONIC) - 1] = '\0';
                                                // THE NEW COMMAND IS PLACED AT THE TOP OF THE LIST
                                                        TMP->NEXT = HANDLE->MNEM_LIST;
                                                        HANDLE->MNEM_LIST = TMP;
                                                }
                                                RETURN CS_ERR_OK;
                                        } ELSE {
                                                STRUCT INSN_MNEM *PREV, *TMP;

                                                TMP = HANDLE->MNEM_LIST;
                                                PREV = TMP;
                                                WHILE(TMP) {
                                                        IF (TMP->INSN.ID == OPT->ID) {
                                                                // DELETE INSTRUCTION
                                                                IF (TMP == PREV) {
                                                                        HANDLE->MNEM_LIST =
TMP->NEXT;
                                                                } ELSE {
                                                                        PREV->NEXT = TMP-
>NEXT;
                                                                }
                                                                CS_MEM_FREE(TMP);
                                                                BREAK;
                                                        }
                                                        PREV = TMP;
                                                        TMP = TMP->NEXT;
                                                }
                                        }
                                }
                                RETURN CS_ERR_OK;

                        CASE CS_OPT_MODE:
                                // VERIFY THAT THE REQUESTED PATTERN IS VALID
                                IF (VALUE & CS_ARCH_DISALLOWED_MODE_MASK[HANDLE->ARCH]) {
                                        RETURN CS_ERR_OPTION;
                                }
                                BREAK;
                }

                RETURN CS_ARCH_OPTION[HANDLE->ARCH](HANDLE, TYPE, VALUE);
}
```

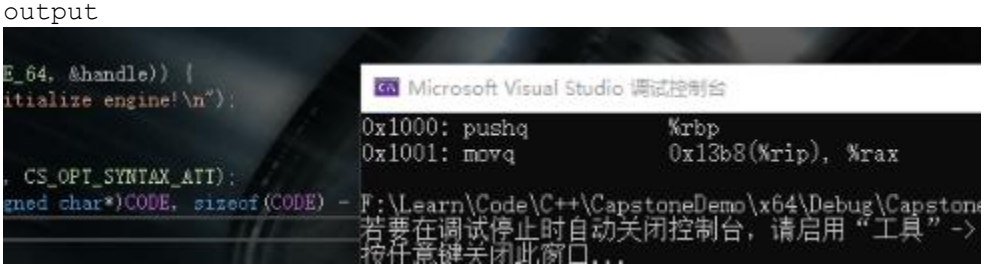
Example, change the syntax displayed after disassembly

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

#DEFINE CODE "\X55\X48\X8B\X05\XB8\X13\X00\X00"

INT MAIN(VOID)
{
        CSH HANDLE;
        CS_INSN* INSN;
        SIZE_T COUNT;

        IF (CS_OPEN(CS_ARCH_X86, CS_MODE_64, &HANDLE)) {
                PRINTF("ERROR: FAILED TO INITIALIZE ENGINE!\N");
                RETURN -1;
        }
        CS_OPTION(HANDLE, CS_OPT_SYNTAX, CS_OPT_SYNTAX_ATT);  // DISPLAYED IN AT&T SYNTAX
        COUNT = CS_DISASM(HANDLE, (UNSIGNED CHAR*)CODE, SIZEOF(CODE) - 1, 0X1000, 0, &INSN);
```

```
        IF (COUNT) {
                SIZE_T J;

                FOR (J = 0; J < COUNT; J++) {
                        PRINTF("0X%""IX"":\T%S\T\T%S\N", INSN[J].ADDRESS, INSN[J].MNEMONIC,
INSN[J].OP_STR);
                }

                CS_FREE(INSN, COUNT);
        }
        ELSE
                PRINTF("ERROR: FAILED TO DISASSEMBLE GIVEN CODE!\N");

        CS_CLOSE(&HANDLE);

        RETURN 0;
}
```

output



cs_errno

An error message is returned when the API makes an error

Handle: cs_open() Open handle

Return: CS_ERR_OK is returned without error, otherwise the error message of cs_err_ok is returned

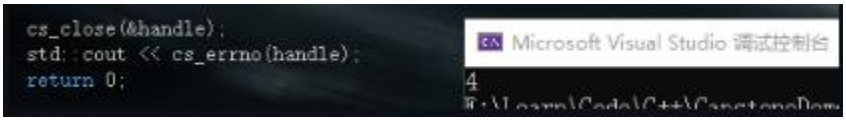
Judge that the handle does not exist and return CS_ERR_CSH directly

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

#DEFINE CODE "\X55\X48\X8B\X05\XB8\X13\X00\X00"

INT MAIN(VOID)
{
        CSH HANDLE = 0;
        CS_INSN* INSN;
        SIZE_T COUNT;

        IF (CS_OPEN(CS_ARCH_X86, CS_MODE_64, &HANDLE)) {
```

```
                printf("ERROR: Failed to initialize engine!\n");
                return -1;
        }

        cs_close(&handle);
        std::cout << cs_errno(handle); // After closing the handle, the check will report an error
        return 0;
}
```

Output, error code 4 is CS_ERR_CSH



cs_sterror

```
const char * CAPSTONE_API cs_strerror(cs_err code);
```

Convert the error code output from the previous API into a detailed error message

Code:

```
const char * CAPSTONE_API cs_strerror(cs_err code)
{
        switch(code) {
                default:
                        return "Unknown error code";
                case CS_ERR_OK:
                        return "OK (CS_ERR_OK)";
                case CS_ERR_MEM:
                        return "Out of memory (CS_ERR_MEM)";
                case CS_ERR_ARCH:
                        return "Invalid/unsupported architecture(CS_ERR_ARCH)";
                case CS_ERR_HANDLE:
                        return "Invalid handle (CS_ERR_HANDLE)";
                case CS_ERR_CSH:
                        return "Invalid csh (CS_ERR_CSH)";
                case CS_ERR_MODE:
                        return "Invalid mode (CS_ERR_MODE)";
                case CS_ERR_OPTION:
                        return "Invalid option (CS_ERR_OPTION)";
                case CS_ERR_DETAIL:
```

```
                        RETURN "DETAILS ARE UNAVAILABLE (CS_ERR_DETAIL)";
                CASE CS_ERR_MEMSETUP:
                        RETURN "DYNAMIC MEMORY MANAGEMENT UNINITIALIZED (CS_ERR_MEMSETUP)";
                CASE CS_ERR_VERSION:
                        RETURN "DIFFERENT API VERSION BETWEEN CORE & BINDING
(CS_ERR_VERSION)";
                CASE CS_ERR_DIET:
                        RETURN "INFORMATION IRRELEVANT IN DIET ENGINE (CS_ERR_DIET)";
                CASE CS_ERR_SKIPDATA:
                        RETURN "INFORMATION IRRELEVANT FOR 'DATA' INSTRUCTION IN SKIPDATA MODE
(CS_ERR_SKIPDATA)";
                CASE CS_ERR_X86_ATT:
                        RETURN "AT&T SYNTAX IS UNAVAILABLE (CS_ERR_X86_ATT)";
                CASE CS_ERR_X86_INTEL:
                        RETURN "INTEL SYNTAX IS UNAVAILABLE (CS_ERR_X86_INTEL)";
                CASE CS_ERR_X86_MASM:
                        RETURN "MASM SYNTAX IS UNAVAILABLE (CS_ERR_X86_MASM)";
        }
}
```

Example, used in combination with cs_errno:

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

#DEFINE CODE "\X55\X48\X8B\X05\XB8\X13\X00\X00"

INT MAIN(VOID)
{
        CSH HANDLE = 0;
        CS_INSN* INSN;
        SIZE_T COUNT;

        IF (CS_OPEN(CS_ARCH_X86, CS_MODE_64, &HANDLE)) {
                PRINTF("ERROR: FAILED TO INITIALIZE ENGINE!\N");
                RETURN -1;
        }

        CS_CLOSE(&HANDLE);
        STD::COUT << CS_STRERROR(CS_ERRNO(HANDLE));  //  DIRECTLY OUTPUT ERROR MESSAGE
        RETURN 0;
}
```

ouput

```
cs_close(&handle);
std::cout << cs_strerror(cs_errno(handle));
return 0;
```

```
Microsoft Visual Studio 调试控制台
Invalid csh (CS_ERR_CSH)
```

cs_disasm

```
size_t CAPSTONE_API cs_disasm(csh handle,onst uint8_t *code, size_t code_size,
uint64_t address, size_t count, cs_insn **insn);
```

Decompile the machine code given the buffer, size, address, and number

The API dynamically allocates memory to contain decomposed instructions, and the generated instructions will be placed in *insn

Note: The allocated memory must be released to avoid memory leaks.For systems that need to dynamically allocate scarce memory (such as OS kernel or firmware), API cs_disasm_iter() may be a better choice than cs_disasm().The reason is that when using cs_disasm(), based on the limited available memory, you must calculate in advance how many instructions to decompose.

Handle: The handle returned by cs_open()
Code: A buffer containing the machine code to be disassembled.

code_size: The size of the code buffer above.

Address: The address of the first instruction in the given original code buffer.

insn: An array of instructions filled in by this API.Note: insn will be allocated by this function and should be released with the cs_free () API

count: The number of instructions that need to be decomposed, or enter 0 to decompose all instructions

Return: The number of instructions successfully disassembled. If the function fails to disassemble the given code, it is 0. When it fails, call cs_errno() to get the error code.

Code:

```
SIZE_T CAPSTONE_API CS_DISASM(CSH UD, CONST UINT8_T *BUFFER, SIZE_T SIZE, UINT64_T OFFSET, SIZE_T
COUNT, CS_INSN **INSN)
{
        STRUCT CS_STRUCT *HANDLE;
        MCINST MCI;
        UINT16_T INSN_SIZE;
        SIZE_T C = 0, I;
        UNSIGNED INT F = 0; // INDEX OF THE NEXT INSTRUCTION IN THE CACHE
        CS_INSN *INSN_CACHE;          // CACHE DISASSEMBLED INSTRUCTIONS
        VOID *TOTAL = NULL;
        SIZE_T TOTAL_SIZE = 0;        // THE TOTAL SIZE OF THE OUTPUT BUFFER OF ALL INSNS
        BOOL R;
        VOID *TMP;
        SIZE_T SKIPDATA_BYTES;
        UINT64_T OFFSET_ORG; // SAVE ALL THE ORIGINAL INFORMATION OF THE BUFFER
        SIZE_T SIZE_ORG;
```

```
        const uint8_t *buffer_org;
        unsigned int cache_size = INSN_CACHE_SIZE;
        size_t next_offset;

        handle = (struct cs_struct *)(uintptr_t)ud;
        if (!handle) {
                // Repair method:
                // handle->errnum = CS_ERR_HANDLE;
                return 0;
        }

        handle->errnum = CS_ERR_OK;

        // Reset the IT block of ARM architecture
        if (handle->arch == CS_ARCH_ARM)
                handle->ITBlock.size = 0;

#ifdef CAPSTONE_USE_SYS_DYN_MEM
        if (count > 0 && count <= INSN_CACHE_SIZE)
                cache_size = (unsigned int) count;
#endif

        // Save the original offset of SKIPDATA
        buffer_org = buffer;
        offset_org = offset;
        size_org = size;

        total_size = sizeof(cs_insn) * cache_size;
        total = cs_mem_malloc(total_size);
        if (total == NULL) {
                // Insufficient memory
                handle->errnum = CS_ERR_MEM;
                return 0;
        }

        insn_cache = total;

        while (size > 0) {
                MCInst_Init(&mci);
                mci.csh = handle;

                mci.address = offset;

                if (handle->detail) {
                        // Allocate memory to the detail pointer
                        insn_cache->detail = cs_mem_malloc(sizeof(cs_detail));
                } else {
                        insn_cache->detail = NULL;
                }

                // Save all information for non-detailed mode
                mci.flat_insn = insn_cache;
```

```
                MCI.FLAT_INSN->ADDRESS = OFFSET;
#IFDEF CAPSTONE_DIET
                // MNEMONIC & OP_STR0 FILL
                MCI.FLAT_INSN->MNEMONIC[0] = '\0';
                MCI.FLAT_INSN->OP_STR[0] = '\0';
#ENDIF

                R = HANDLE->DISASM(UD, BUFFER, SIZE, &MCI, &INSN_SIZE, OFFSET, HANDLE-
>GETINSN_INFO);
                IF (R) {
                        SSTREAM SS;
                        SSTREAM_INIT(&SS);

                        MCI.FLAT_INSN->SIZE = INSN_SIZE;

                        // MAP INTERNAL INSTRUCTION OPCODES TO PUBLIC INSN IDS
                        HANDLE->INSN_ID(HANDLE, INSN_CACHE, MCI.OPCODE);

                        HANDLE->PRINTER(&MCI, &SS, HANDLE->PRINTER_INFO);
                        FILL_INSN(HANDLE, INSN_CACHE, SS.BUFFER, &MCI, HANDLE->POST_PRINTER,
BUFFER);

                        // ADJUST OPCODE (X86)
                        IF (HANDLE->ARCH == CS_ARCH_X86)
                                INSN_CACHE->ID += MCI.POPCODE_ADJUST;

                        NEXT_OFFSET = INSN_SIZE;
                } ELSE  {
                        // ENCOUNTERED AN INTERRUPT COMMAND

                        // FREE UP MEMORY FOR THE DETAIL POINTER
                        IF (HANDLE->DETAIL) {
                                CS_MEM_FREE(INSN_CACHE->DETAIL);
                        }

                        IF (!HANDLE->SKIPDATA || HANDLE->SKIPDATA_SIZE > SIZE)
                                BREAK;

                        IF (HANDLE->SKIPDATA_SETUP.CALLBACK) {
                                SKIPDATA_BYTES = HANDLE->SKIPDATA_SETUP.CALLBACK(BUFFER_ORG,
SIZE_ORG,
                                                (SIZE_T)(OFFSET - OFFSET_ORG), HANDLE-
>SKIPDATA_SETUP.USER_DATA);
                                IF (SKIPDATA_BYTES > SIZE)
                                        BREAK;

                                IF (!SKIPDATA_BYTES)
                                        BREAK;
                        } ELSE
                                SKIPDATA_BYTES = HANDLE->SKIPDATA_SIZE;

                        INSN_CACHE->ID = 0;
                        INSN_CACHE->ADDRESS = OFFSET;
```

```
                        INSN_CACHE->SIZE = (UINT16_T)SKIPDATA_BYTES;
                        MEMCPY(INSN_CACHE->BYTES, BUFFER, SKIPDATA_BYTES);
#IFDEF CAPSTONE_DIET
                        INSN_CACHE->MNEMONIC[0] = '\0';
                        INSN_CACHE->OP_STR[0] = '\0';
#ELSE
                        STRNCPY(INSN_CACHE->MNEMONIC, HANDLE->SKIPDATA_SETUP.MNEMONIC,
                                        SIZEOF(INSN_CACHE->MNEMONIC) - 1);
                        SKIPDATA_OPSTR(INSN_CACHE->OP_STR, BUFFER, SKIPDATA_BYTES);
#ENDIF
                        INSN_CACHE->DETAIL = NULL;

                        NEXT_OFFSET = SKIPDATA_BYTES;
                }

                // A NEW INSTRUCTION ENTERS THE CACHE
                F++;

                // DISASSEMBLED AN INSTRUCTION
                C++;
                IF (COUNT > 0 && C == COUNT)
                        BREAK;

                IF (F == CACHE_SIZE) {
                        CACHE_SIZE = CACHE_SIZE * 8 / 5;
                        TOTAL_SIZE += (SIZEOF(CS_INSN) * CACHE_SIZE);
                        TMP = CS_MEM_REALLOC(TOTAL, TOTAL_SIZE);
                        IF (TMP == NULL) { //内存不足
                                IF (HANDLE->DETAIL) {
                                        INSN_CACHE = (CS_INSN *)TOTAL;
                                        FOR (I = 0; I < C; I++, INSN_CACHE++)
                                                CS_MEM_FREE(INSN_CACHE->DETAIL);
                                }

                                CS_MEM_FREE(TOTAL);
                                *INSN = NULL;
                                HANDLE->ERRNUM = CS_ERR_MEM;
                                RETURN 0;
                        }

                        TOTAL = TMP;
                        // CONTINUE TO FILL THE CACHE AFTER THE LAST INSTRUCTION
                        INSN_CACHE = (CS_INSN *)((CHAR *)TOTAL + SIZEOF(CS_INSN) * C);

                        // RESET F TO 0 AND FILL IN THE CACHE FROM THE BEGINNING
                        F = 0;
                } ELSE
                        INSN_CACHE++;

                BUFFER += NEXT_OFFSET;
                SIZE -= NEXT_OFFSET;
                OFFSET += NEXT_OFFSET;
```

```
        }

        if (!c) {
                // No instructions have been disassembled
                cs_mem_free(total);
                total = NULL;
        } else if (f != cache_size) {
                // Did not fully use the last cache, reduce the size
                tmp = cs_mem_realloc(total, total_size - (cache_size - f) * sizeof(*insn_cache));
                if (tmp == NULL) { // Insufficient memory
                        // Release all detail pointers
                        if (handle->detail) {
                                insn_cache = (cs_insn *)total;
                                for (i = 0; i < c; i++, insn_cache++)
                                        cs_mem_free(insn_cache->detail);
                        }

                        cs_mem_free(total);
                        *insn = NULL;

                        handle->errnum = cs_err_mem;
                        return 0;
                }

                total = tmp;
        }

        *insn = total;

        return c;
}
```

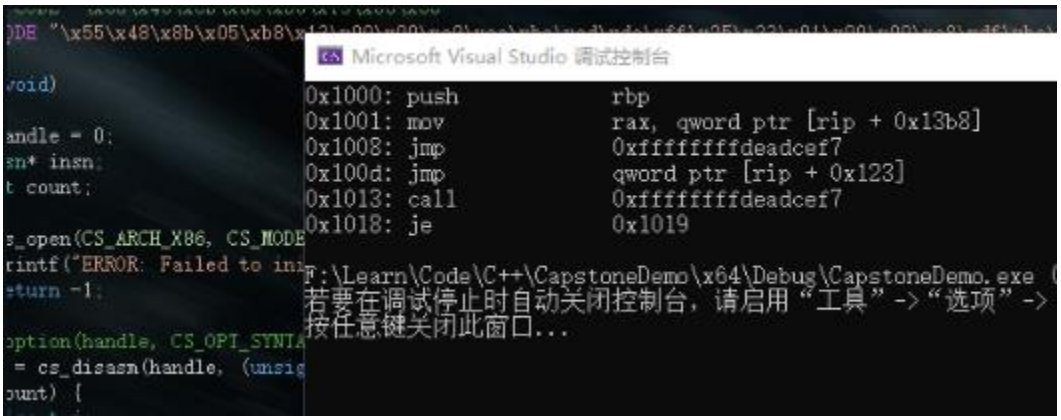
Example, x86_64

Code:

```
#include <iostream>
#include <stdio.h>

#include "capstone.h"
#include "platform.h"

using namespace std;

#define code
"\x55\x48\x8b\x05\xb8\x13\x00\x00\xe9\xea\xbe\xad\xde\xff\x25\x23\x01\x00\x00\xe8\xdf\xbe\xad
\xde\x74\xff"
```

```
int main(void)
{
        csh handle = 0;
        cs_insn* insn;
        size_t count;

        if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle)) {
                printf("ERROR: Failed to initialize engine!\n");
                return -1;
        }

        count = cs_disasm(handle, (unsigned char*)CODE, sizeof(CODE) - 1, 0x1000, 0, &insn);
// All instructions, base address 0X1000, put in INSN
        if (count) {
                size_t j;

                for (j = 0; j < count; j++) {
                        printf("0x%""Ix"":\t%s\t\t%s\n", insn[j].address, insn[j].mnemonic,
insn[j].op_str);
                }

                cs_free(insn, count);
        }
        else
                printf("ERROR: Failed to disassemble given code!\n");

        cs_close(&handle);

        return 0;
}
```

output



cs_free

```
void CAPSTONE_API cs_free(cs_insn *insn, size_t count);
```

Release the memory allocated by cs_malloc() or cs_disasm() (insn parameter)

insn: Pointer returned by the @insn parameter in cs_disasm() or cs_malloc()

count: Assign the number of cs_insn structures returned by cs_disasm(), or assign a value of 1 to indicate the number of free memory allocated by cs_malloc()

Code:

```
VOID CAPSTONE_API CS_FREE(CS_INSN *INSN, SIZE_T COUNT)
{
        SIZE_T I;

        // FREE 所有 DETAIL 指针
        FOR (I = 0; I < COUNT; I++)
                CS_MEM_FREE(INSN[I].DETAIL);

        CS_MEM_FREE(INSN);
}
```

Directly call cs_mem_free, which is the default free

Example (free up the memory requested by cs_disasm)

Code:

```
count = cs_disasm(handle, (unsigned char*)code, sizeof(code) - 1, 0x1000, 0, &insn); // Count the memory requested by cs_disasm
        if (count) {
                size_t j;

                for (j = 0; j < count; j++) {
                        printf("0x%""Ix"":\t%s\t\t%s\n", insn[j].address, insn[j].mnemonic, insn[j].op_str);
                }

                cs_free(insn, count);   // The loop releases the memory of each insn in turn
        }
```

cs_malloc

```
cs_insn * CAPSTONE_API cs_malloc(csh handle);
```

Is used to allocate memory for an instruction in API cs_disasm_iter()

Handle: The handle returned by cs_open()

Code:

```
cs_insn * CAPSTONE_API cs_malloc(csh ud)
{
        cs_insn *insn;
        struct cs_struct *handle = (struct cs_struct *)(uintptr_t)ud;

        insn = cs_mem_malloc(sizeof(cs_insn));
        if (!insn) {
                // insufficient memory
                handle->errnum = CS_ERR_MEM;
                return NULL;
        } else {
                if (handle->detail) {
                        // allocate memory for @detail pointer
                        insn->detail = cs_mem_malloc(sizeof(cs_detail));
                        if (insn->detail == NULL) {  // insufficient memory
                                cs_mem_free(insn);
                                handle->errnum = CS_ERR_MEM;
                                return NULL;
                        }
                } else
                        insn->detail = NULL;
        }

        return insn;
}
```

When the memory occupied by this instruction is no longer used, use cs_free (insn, 1) to release it. The example is at cs_disasm_iter below.

cs_disasm_iter

```
bool CAPSTONE_API cs_disasm_iter(csh handle,const uint8_t **code, size_t *size,
uint64_t *address, cs_insn *insn);
```

Given the buff, size, address, and number of instructions to be decoded, to disassemble the machine code faster, this API puts the generated instructions into the given cache in insn.

Note 1: This API will update the code, size, and address to point to the next instruction in the input buffer.Therefore, although disassembling one instruction at a time can be achieved using cs_disasm(count=1), some benchmark tests show that using cs_disasm_iter() in a loop can easily iterate over all instructions quickly, and it can be 30% faster when randomly entered.

Note 2: You can use cs_malloc() to create a cache in insn.

Note 3: For systems that dynamically allocate memory that may produce insufficient memory (such as OS kernel or firmware), it is recommended to use the cs_disasm () API, because cs_disasm () allocates memory based on the number of instructions to be decomposed.

Handle: The handle returned by cs_open()

Code: The buffer where the machine code to be disassembled is located

Size: The size of the machine code buffer

Address: The address of the first insn in the given machine code buffer

insn: A pointer to the instruction to be populated by this API.

Return: If this API successfully disassembles an instruction, it returns true, otherwise it will return false.

When it fails, call cs_errno() to get the error code.

Code implementation, using dynamic memory allocation on the basis of cs_disasm

Code:

```
BOOL CAPSTONE_API CS_DISASM_ITER(CSH UD, CONST UINT8_T **CODE, SIZE_T *SIZE,
                  UINT64_T *ADDRESS, CS_INSN *INSN)
{
        STRUCT CS_STRUCT *HANDLE;
        UINT16_T INSN_SIZE;
        MCINST MCI;
        BOOL R;

        HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;
        IF (!HANDLE) {
                RETURN FALSE;
        }

        HANDLE->ERRNUM = CS_ERR_OK;

        MCINST_INIT(&MCI);
        MCI.CSH = HANDLE;

        MCI.ADDRESS = *ADDRESS;

        // SAVE RELEVANT INFORMATION FOR NO DETAIL MODE
        MCI.FLAT_INSN = INSN;
        MCI.FLAT_INSN->ADDRESS = *ADDRESS;
#IFDEF CAPSTONE_DIET
        MCI.FLAT_INSN->MNEMONIC[0] = '\0';
        MCI.FLAT_INSN->OP_STR[0] = '\0';
#ENDIF

        R = HANDLE->DISASM(UD, *CODE, *SIZE, &MCI, &INSN_SIZE, *ADDRESS, HANDLE->GETINSN_INFO);
        IF (R) {
                SSTREAM SS;
                SSTREAM_INIT(&SS);

                MCI.FLAT_INSN->SIZE = INSN_SIZE;

                // MAP INTERNAL INSTRUCTION OPCODES TO PUBLIC INSN IDS
                HANDLE->INSN_ID(HANDLE, INSN, MCI.OPCODE);

                HANDLE->PRINTER(&MCI, &SS, HANDLE->PRINTER_INFO);

                FILL_INSN(HANDLE, INSN, SS.BUFFER, &MCI, HANDLE->POST_PRINTER, *CODE);

                // ADJUST PSEUDO-OPCODE (X86)
                IF (HANDLE->ARCH == CS_ARCH_X86)
                        INSN->ID += MCI.POPCODE_ADJUST;

                *CODE += INSN_SIZE;
                *SIZE -= INSN_SIZE;
                *ADDRESS += INSN_SIZE;
        } ELSE {  // ENCOUNTERED AN INTERRUPT COMMAND
                SIZE_T SKIPDATA_BYTES;
```

```
                // If there is no request to skip the data, or the remaining data is too small, then exit
                if (!handle->skipdata || handle->skipdata_size > *size)
                        return false;

                if (handle->skipdata_setup.callback) {
                        skipdata_bytes = handle->skipdata_setup.callback(*code, *size,
                                                0, handle->skipdata_setup.user_data);
                        if (skipdata_bytes > *size)
                                // The remaining data is too small
                                return false;

                        if (!skipdata_bytes)
                                return false;
                } else
                        skipdata_bytes = handle->skipdata_size;

                // Skip some data based on architecture and pattern
                insn->id = 0;       // The ID of this "data" instruction is invalid
                insn->address = *address;
                insn->size = (uint16_t)skipdata_bytes;
#ifdef capstone_diet
                insn->mnemonic[0] = '\0';
                insn->op_str[0] = '\0';
#else
                memcpy(insn->bytes, *code, skipdata_bytes);
                strncpy(insn->mnemonic, handle->skipdata_setup.mnemonic,
                                sizeof(insn->mnemonic) - 1);
                skipdata_opstr(insn->op_str, *code, skipdata_bytes);
#endif

                *code += skipdata_bytes;
                *size -= skipdata_bytes;
                *address += skipdata_bytes;
        }

        return true;
}
```

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

STRUCT PLATFORM {
          CS_ARCH ARCH;
          CS_MODE MODE;
          UNSIGNED CHAR* CODE;
          SIZE_T SIZE;
          CONST CHAR* COMMENT;
          CS_OPT_TYPE OPT_TYPE;
          CS_OPT_VALUE OPT_VALUE;
};

STATIC VOID PRINT_STRING_HEX(UNSIGNED CHAR* STR, SIZE_T LEN)
{
          UNSIGNED CHAR* C;

          PRINTF("CODE: ");
          FOR (C = STR; C < STR + LEN; C++) {
                    PRINTF("0X%02X ", *C & 0XFF);
          }
          PRINTF("\N");
}

STATIC VOID TEST()
{
#DEFINE X86_CODE16 "\X8D\X4C\X32\X08\X01\XD8\X81\XC6\X34\X12\X00\X00"
#DEFINE X86_CODE32 "\X8D\X4C\X32\X08\X01\XD8\X81\XC6\X34\X12\X00\X00"
#DEFINE X86_CODE64 "\X55\X48\X8B\X05\XB8\X13\X00\X00"

          STRUCT PLATFORM PLATFORMS[4] = {     //ARCHITECTURE AND MODE
                    {
                              CS_ARCH_X86,
                              CS_MODE_16,
                              (UNSIGNED CHAR*)X86_CODE16,
                              SIZEOF(X86_CODE32) - 1,
                              "X86 16BIT (INTEL SYNTAX)"
                    },
                    {
                              CS_ARCH_X86,
                              CS_MODE_32,
                              (UNSIGNED CHAR*)X86_CODE32,
                              SIZEOF(X86_CODE32) - 1,
                              "X86 32BIT (ATT SYNTAX)",
                              CS_OPT_SYNTAX,
                              CS_OPT_SYNTAX_ATT,
                    },
```

```
		{
			CS_ARCH_X86,
			CS_MODE_32,
			(unsigned char*)X86_CODE32,
			sizeof(X86_CODE32) - 1,
			"X86 32 (Intel syntax)"
		},
		{
			CS_ARCH_X86,
			CS_MODE_64,
			(unsigned char*)X86_CODE64,
			sizeof(X86_CODE64) - 1,
			"X86 64 (Intel syntax)"
		},

	csh handle;
	uint64_t address;
	cs_insn* insn;
	cs_detail* detail;
	int i;
	cs_err err;
	const uint8_t* code;
	size_t size;

	for (i = 0; i < sizeof(platforms) / sizeof(platforms[0]); i++) {
		printf("****************\n");
		printf("Platform: %s\n", platforms[i].comment);
		err = cs_open(platforms[i].arch, platforms[i].mode, &handle);
		if (err) {
			printf("Failed on cs_open() with error returned: %u\n", err);
			abort();
		}

		if (platforms[i].opt_type)
			cs_option(handle, platforms[i].opt_type, platforms[i].opt_value);

		cs_option(handle, CS_OPT_DETAIL, CS_OPT_ON);

		// Allocate memory for CS_DISASM_ITER()
		insn = cs_malloc(handle);

		print_string_hex(platforms[i].code, platforms[i].size);	// Original machine
code
		printf("Disasm:\n");

		address = 0x1000;
		code = platforms[i].code;
		size = platforms[i].size;
		while (cs_disasm_iter(handle, &code, &size, &address, insn)) {//CS_DISASM_ITER
Disassembly
			int n;

			printf("0x%" PRIx64 ":\t%s\t\t%s // insn-ID: %u, insn-mnem: %s\n",
				insn->address, insn->mnemonic, insn->op_str,
```

```
                                        INSN->ID, CS_INSN_NAME(HANDLE, INSN->ID));

                                // Print the implicit register used by this command
                                detail = insn->detail;

                                if (detail->regs_read_count > 0) {
                                        printf("\tImplicit registers read: ");
                                        for (n = 0; n < detail->regs_read_count; n++) {
                                                printf("%s ", cs_reg_name(handle, detail-
>regs_read[n]));
                                        }
                                        printf("\n");
                                }

                                // Print the implicit register modified by this command
                                if (detail->regs_write_count > 0) {
                                        printf("\tImplicit registers modified: ");
                                        for (n = 0; n < detail->regs_write_count; n++) {
                                                printf("%s ", cs_reg_name(handle, detail-
>regs_write[n]));
                                        }
                                        printf("\n");
                                }

                                // Print the instruction set to which this instruction belongs
                                if (detail->groups_count > 0) {
                                        printf("\tThis instruction belongs to groups: ");
                                        for (n = 0; n < detail->groups_count; n++) {
                                                printf("%s ", cs_group_name(handle, detail-
>groups[n]));
                                        }
                                        printf("\n");
                                }
                        }

                        printf("\n");

                        // Free up the memory allocated by cs_malloc()
                        cs_free(insn, 1);

                        cs_close(&handle);
                }
}

int main()
{
        test();

        return 0;
}
```

output

```
Microsoft Visual Studio 调试控制台
****************
Platform: X86 16bit (Intel syntax)
Code: 0x8d 0x4c 0x32 0x08 0x01 0xd8 0x81 0xc6 0x34 0x12 0x00 0x00
Disasm:
0x1000: lea             cx, [si + 0x32] // insn-ID: 322, insn-mnem: lea
0x1003: or              byte ptr [bx + di], al // insn-ID: 332, insn-mnem: or
        Implicit registers modified: flags
0x1005: fadd            dword ptr [bx + di + 0x34c6] // insn-ID: 15, insn-mnem: fadd
        Implicit registers modified: fpsw
        This instruction belongs to groups: fpu
0x1009: adc             al, byte ptr [bx + si] // insn-ID: 6, insn-mnem: adc
        Implicit registers read: flags
        Implicit registers modified: flags

****************
Platform: X86 32bit (ATT syntax)
Code: 0x8d 0x4c 0x32 0x08 0x01 0xd8 0x81 0xc6 0x34 0x12 0x00 0x00
Disasm:
0x1000: leal            8(%edx, %esi), %ecx // insn-ID: 322, insn-mnem: lea
        This instruction belongs to groups: not64bitmode
0x1004: addl            %ebx, %eax // insn-ID: 8, insn-mnem: add
        Implicit registers modified: eflags
0x1006: addl            $0x1234, %esi // insn-ID: 8, insn-mnem: add
        Implicit registers modified: eflags

****************
Platform: X86 32 (Intel syntax)
Code: 0x8d 0x4c 0x32 0x08 0x01 0xd8 0x81 0xc6 0x34 0x12 0x00 0x00
Disasm:
0x1000: lea             ecx, [edx + esi + 8] // insn-ID: 322, insn-mnem: lea
        This instruction belongs to groups: not64bitmode
0x1004: add             eax, ebx // insn-ID: 8, insn-mnem: add
        Implicit registers modified: eflags
0x1006: add             esi, 0x1234 // insn-ID: 8, insn-mnem: add
        Implicit registers modified: eflags

****************
Platform: X86 64 (Intel syntax)
Code: 0x55 0x48 0x8b 0x05 0xb8 0x13 0x00 0x00
Disasm:
0x1000: push            rbp // insn-ID: 588, insn-mnem: push
        Implicit registers read: rsp
        Implicit registers modified: rsp
        This instruction belongs to groups: mode64
0x1001: mov             rax, qword ptr [rip + 0x13b8] // insn-ID: 449, insn-mnem: mov

F:\Learn\Code\C++\CapstoneDemo\x64\Debug\CapstoneDemo.exe (进程 14648)已退出，返回代码为：0。
```

cs_reg_name

```
const char * CAPSTONE_API cs_reg_name(csh handle, unsigned int reg_id);
```

Get the name of the register (string type) and the register id can be found in the header files of the relevant architecture (those header files copied to the project folder when the project is created)

Note: This API is not available when in diet mode because the engine does not store register names

Handle: The handle returned by cs_open()

reg_id: register id

Return: The character name of the register, return NULL if reg_id is not available

Code:

```
CONST CHAR * CAPSTONE_API CS_REG_NAME(CSH UD, UNSIGNED INT REG)
{
        STRUCT CS_STRUCT *HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

        IF (!HANDLE || HANDLE->REG_NAME == NULL) {
                RETURN NULL;
        }

        RETURN HANDLE->REG_NAME(UD, REG);
}
```

Example (print RAX)

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

INT MAIN(VOID)
{
        CSH HANDLE = 0;
        CS_INSN* INSN;
        SIZE_T COUNT;

        IF (CS_OPEN(CS_ARCH_X86, CS_MODE_64, &HANDLE)) {
                PRINTF("ERROR: FAILED TO INITIALIZE ENGINE!\N");
                RETURN -1;
        }

        PRINTF("%S", CS_REG_NAME(HANDLE, X86_REG_RAX));
        CS_CLOSE(&HANDLE);

        RETURN 0;
}
```

OUTPUT

cs_insn_name

```
const char * CAPSTONE_API cs_insn_name(csh handle, unsigned int insn_id);
```

Get the name of the instruction (string type)

The instruction id can be found in the header files of the relevant architecture (those header files copied to the project folder when the project is created)

Note: This API is not available when in diet mode because the engine does not store register names

Handle: The handle returned by cs_open()

insn_id: instruction id

return: The character name of the instruction, return NULL if insn_id is not available

Code:

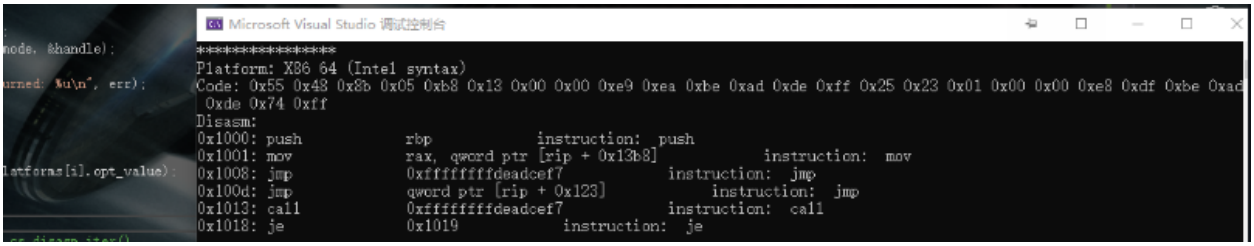
```
CONST CHAR * CAPSTONE_API CS_INSN_NAME(CSH UD, UNSIGNED INT INSN)
{
        STRUCT CS_STRUCT *HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

        IF (!HANDLE || HANDLE->INSN_NAME == NULL) {
                RETURN NULL;
        }

        RETURN HANDLE->INSN_NAME(UD, INSN);
}
```

Code#2:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

STRUCT PLATFORM {
          CS_ARCH ARCH;
          CS_MODE MODE;
          UNSIGNED CHAR* CODE;
          SIZE_T SIZE;
          CONST CHAR* COMMENT;
          CS_OPT_TYPE OPT_TYPE;
          CS_OPT_VALUE OPT_VALUE;
};

STATIC VOID PRINT_STRING_HEX(UNSIGNED CHAR* STR, SIZE_T LEN)
{
          UNSIGNED CHAR* C;

          PRINTF("CODE: ");
          FOR (C = STR; C < STR + LEN; C++) {
                    PRINTF("0X%02X ", *C & 0XFF);
          }
          PRINTF("\N");
}

STATIC VOID TEST()
{

#DEFINE X86_CODE64
"\X55\X48\X8B\X05\XB8\X13\X00\X00\XE9\XEA\XBE\XAD\XDE\XFF\X25\X23\X01\X00\X00\XE8\XDF\XBE\XAD
\XDE\X74\XFF"

          STRUCT PLATFORM PLATFORMS[] = {
                    {
                              CS_ARCH_X86,
                              CS_MODE_64,
                              (UNSIGNED CHAR*)X86_CODE64,
                              SIZEOF(X86_CODE64) - 1,
                              "X86 64 (INTEL SYNTAX)"
                    },
          };

          CSH HANDLE;
          UINT64_T ADDRESS;
          CS_INSN* INSN;
          CS_DETAIL* DETAIL;
          INT I;
          CS_ERR ERR;
          CONST UINT8_T* CODE;
```

```
        SIZE_T SIZE;

        FOR (I = 0; I < SIZEOF(PLATFORMS) / SIZEOF(PLATFORMS[0]); I++) {
                PRINTF("****************\N");
                PRINTF("PLATFORM: %S\N", PLATFORMS[I].COMMENT);
                ERR = CS_OPEN(PLATFORMS[I].ARCH, PLATFORMS[I].MODE, &HANDLE);
                IF (ERR) {
                        PRINTF("FAILED ON CS_OPEN() WITH ERROR RETURNED: %U\N", ERR);
                        ABORT();
                }

                IF (PLATFORMS[I].OPT_TYPE)
                        CS_OPTION(HANDLE, PLATFORMS[I].OPT_TYPE, PLATFORMS[I].OPT_VALUE);

                CS_OPTION(HANDLE, CS_OPT_DETAIL, CS_OPT_ON);

                INSN = CS_MALLOC(HANDLE);

                PRINT_STRING_HEX(PLATFORMS[I].CODE, PLATFORMS[I].SIZE);
                PRINTF("DISASM:\N");

                ADDRESS = 0X1000;
                CODE = PLATFORMS[I].CODE;
                SIZE = PLATFORMS[I].SIZE;
                WHILE (CS_DISASM_ITER(HANDLE, &CODE, &SIZE, &ADDRESS, INSN)) {
                        INT N;

                        PRINTF("0X%" PRIX64 ":\T%S\T\T%S",
                                INSN->ADDRESS, INSN->MNEMONIC, INSN->OP_STR);
                        PRINTF("            INSTRUCTION:  %S", CS_INSN_NAME(HANDLE, INSN->ID));
// OUTPUT THE OPERATION INSTRUCTION OF THE LINE
                        COUT << ENDL;

                PRINTF("\N");
                CS_FREE(INSN, 1);
                CS_CLOSE(&HANDLE);
        }
}

INT MAIN()
{
        TEST();

        RETURN 0;
}
```

output



cs_group_name

const char * CAPSTONE_API cs_group_name(csh handle, unsigned int group_id);

Output instruction type name

The instruction id can be found in the header files of the relevant architecture (those header files copied to the project folder when the project is created)

Note: This API is not available when in diet mode because the engine does not store register names

Handle: The handle returned by cs_open()

insn_id: instruction type id

return: The character name of the instruction type, return NULL if insn_id is not available

The examples are similar to the above, slightly.

cs_insn_group

```
bool CAPSTONE_API cs_insn_group(csh handle, const cs_insn *insn, unsigned int
group_id);
```

Check whether the disassembled instruction belongs to a specific instruction type

Note: This API is only available when the detail option is ON (OFF by default).

In "diet" mode, this API is useless because the engine does not update the insn->groups array

Handle: The handle returned by cs_open()

insn: Disassembly instruction structure received from cs_disasm() or cs_disasm_iter()

group_id: The type of instruction to check whether this instruction belongs to.

Return: True if the instruction does belong to the given instruction type, false otherwise.

Code:

```
BOOL CAPSTONE_API CS_INSN_GROUP(CSH UD, CONST CS_INSN *INSN, UNSIGNED INT GROUP_ID)
{
        STRUCT CS_STRUCT *HANDLE;
        IF (!UD)
                RETURN FALSE;

        HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

        IF (!HANDLE->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN FALSE;
        }

        IF (!INSN->ID) {
                HANDLE->ERRNUM = CS_ERR_SKIPDATA;
                RETURN FALSE;
        }

        IF (!INSN->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN FALSE;
        }

        RETURN ARR_EXIST8(INSN->DETAIL->GROUPS, INSN->DETAIL->GROUPS_COUNT, GROUP_ID);
}
```

Example (to determine whether it belongs to a jump instruction)

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

STRUCT PLATFORM {
          CS_ARCH ARCH;
          CS_MODE MODE;
          UNSIGNED CHAR* CODE;
          SIZE_T SIZE;
          CONST CHAR* COMMENT;
          CS_OPT_TYPE OPT_TYPE;
          CS_OPT_VALUE OPT_VALUE;
};

STATIC VOID PRINT_STRING_HEX(UNSIGNED CHAR* STR, SIZE_T LEN)
{
          UNSIGNED CHAR* C;

          PRINTF("CODE: ");
          FOR (C = STR; C < STR + LEN; C++) {
                    PRINTF("0X%02X ", *C & 0XFF);
          }
          PRINTF("\N");
}

STATIC VOID TEST()
{

#DEFINE X86_CODE64
"\X55\X48\X8B\X05\XB8\X13\X00\X00\XE9\XEA\XBE\XAD\XDE\XFF\X25\X23\X01\X00\X00\XE8\XDF\XBE\XAD
\XDE\X74\XFF"

          STRUCT PLATFORM PLATFORMS[] = {
                    {
                              CS_ARCH_X86,
                              CS_MODE_64,
                              (UNSIGNED CHAR*)X86_CODE64,
                              SIZEOF(X86_CODE64) - 1,
                              "X86 64 (INTEL SYNTAX)"
                    },
          };

          CSH HANDLE;
          UINT64_T ADDRESS;
          CS_INSN* INSN;
          CS_DETAIL* DETAIL;
          INT I;
          CS_ERR ERR;
          CONST UINT8_T* CODE;
```

```
        SIZE_T SIZE;

        FOR (I = 0; I < SIZEOF(PLATFORMS) / SIZEOF(PLATFORMS[0]); I++) {
                PRINTF("****************\N");
                PRINTF("PLATFORM: %S\N", PLATFORMS[I].COMMENT);
                ERR = CS_OPEN(PLATFORMS[I].ARCH, PLATFORMS[I].MODE, &HANDLE);
                IF (ERR) {
                        PRINTF("FAILED ON CS_OPEN() WITH ERROR RETURNED: %U\N", ERR);
                        ABORT();
                }

                IF (PLATFORMS[I].OPT_TYPE)
                        CS_OPTION(HANDLE, PLATFORMS[I].OPT_TYPE, PLATFORMS[I].OPT_VALUE);

                CS_OPTION(HANDLE, CS_OPT_DETAIL, CS_OPT_ON);

                INSN = CS_MALLOC(HANDLE);

                PRINT_STRING_HEX(PLATFORMS[I].CODE, PLATFORMS[I].SIZE);
                PRINTF("DISASM:\N");

                ADDRESS = 0X1000;
                CODE = PLATFORMS[I].CODE;
                SIZE = PLATFORMS[I].SIZE;
                WHILE (CS_DISASM_ITER(HANDLE, &CODE, &SIZE, &ADDRESS, INSN)) {
                        INT N;

                        PRINTF("0X%" PRIX64 ":\T%S\T\T%S         ",
                                INSN->ADDRESS, INSN->MNEMONIC, INSN->OP_STR);
                        COUT << "IS JUMP:   " <<CS_INSN_GROUP(HANDLE, INSN, CS_GRP_JUMP) <<
ENDL;   // DETERMINE WHETHER IT IS A JUMP COMMAND
                        COUT << ENDL;

                PRINTF("\N");
                CS_FREE(INSN, 1);
                CS_CLOSE(&HANDLE);
        }
}

INT MAIN()
{
        TEST();

        RETURN 0;
}
```

output

```
****************
Platform: X86 64 (Intel syntax)
Code: 0x55 0x48 0x8b 0x05 0xb8 0x13 0x00 0x00 0xe9 0xea 0xbe 0xad 0xde 0xff 0x25 0x23 0x01 0x00 0x00 0xe8 0xdf 0xbe 0xad
 0xde 0x74 0xff
Disasm:
0x1000: push            rbp          is JUMP:   0

0x1001: mov             rax, qword ptr [rip + 0x13b8]         is JUMP:   0

0x1008: jmp             0xffffffffdeadcef7         is JUMP:   1

0x100d: jmp             qword ptr [rip + 0x123]         is JUMP:   1

0x1013: call            0xffffffffdeadcef7         is JUMP:   0

0x1018: je              0x1019         is JUMP:   1
```

cs_reg_read

```
bool CAPSTONE_API cs_reg_read(csh handle, const cs_insn *insn, unsigned int reg_id);
```

Check whether the disassembly instruction implicitly uses a specific register.

Note: This API is only valid when the detail option is enabled (the default is off)

In "diet" mode, this API is useless because the engine does not update the insn->regs_read array

insn: Disassembly instruction structure received from cs_disasm() or cs_disasm_iter()

reg_id: Mark whether the instruction you want to check uses it.

Return: True if the instruction does implicitly use the given register, otherwise false.

Code:

```
BOOL CAPSTONE_API CS_REG_READ(CSH UD, CONST CS_INSN *INSN, UNSIGNED INT REG_ID)
{
        STRUCT CS_STRUCT *HANDLE;
        IF (!UD)
                RETURN FALSE;

        HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

        IF (!HANDLE->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN FALSE;
        }

        IF (!INSN->ID) {
                HANDLE->ERRNUM = CS_ERR_SKIPDATA;
                RETURN FALSE;
        }

        IF (!INSN->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN FALSE;
        }

        RETURN ARR_EXIST(INSN->DETAIL->REGS_READ, INSN->DETAIL->REGS_READ_COUNT, REG_ID);
}
```

The example is the same as API cs_disasm_iter

cs_reg_write

```
bool CAPSTONE_API cs_reg_write(csh handle, const cs_insn *insn, unsigned int reg_id);
```

Check whether the disassembly instruction implicitly modifies a specific register.

Note: This API is only valid when the detail option is enabled (the default is off)

In "diet" mode, this API is useless because the engine does not update the insn->regs_read array

insn: Disassembly instruction structure received from cs_disasm() or cs_disasm_iter()

reg_id: Mark whether the instruction you want to check has modified it.

Return: True if the instruction does implicitly modify the given register, otherwise false.

Code:

```
BOOL CAPSTONE_API CS_REG_WRITE(CSH UD, CONST CS_INSN *INSN, UNSIGNED INT REG_ID)
{
        STRUCT CS_STRUCT *HANDLE;
        IF (!UD)
                RETURN FALSE;

        HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

        IF (!HANDLE->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN FALSE;
        }

        IF (!INSN->ID) {
                HANDLE->ERRNUM = CS_ERR_SKIPDATA;
                RETURN FALSE;
        }

        IF (!INSN->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN FALSE;
        }

        RETURN ARR_EXIST(INSN->DETAIL->REGS_WRITE, INSN->DETAIL->REGS_WRITE_COUNT, REG_ID);
}
```

The example is the same as API cs_disasm_iter

cs_op_count

```
int CAPSTONE_API cs_op_count(csh handle, const cs_insn *insn, unsigned int op_type);
```

Calculate the number of operands of a given type

Note: This API is only available when the detail option is ON (OFF by default).

Handle: The handle returned by cs_open()
insn: Disassembly instruction structure received from cs_disasm() or cs_disasm_iter()
op_type: The type of operand to be found.
Return: The number of operands of the given type op_type in the instruction insn, returning -1 indicates that the lookup failed.

Code:

```
INT CAPSTONE_API CS_OP_COUNT(CSH UD, CONST CS_INSN *INSN, UNSIGNED INT OP_TYPE)
{
        STRUCT CS_STRUCT *HANDLE;
        UNSIGNED INT COUNT = 0, I;
        IF (!UD)
                RETURN -1;

        HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

        IF (!HANDLE->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN -1;
        }

        IF (!INSN->ID) {
                HANDLE->ERRNUM = CS_ERR_SKIPDATA;
                RETURN -1;
        }

        IF (!INSN->DETAIL) {
                HANDLE->ERRNUM = CS_ERR_DETAIL;
                RETURN -1;
        }

        HANDLE->ERRNUM = CS_ERR_OK;

        SWITCH (HANDLE->ARCH) {
                DEFAULT:
                        HANDLE->ERRNUM = CS_ERR_HANDLE;
                        RETURN -1;
                CASE CS_ARCH_ARM:
                        FOR (I = 0; I < INSN->DETAIL->ARM.OP_COUNT; I++)
                                IF (INSN->DETAIL->ARM.OPERANDS[I].TYPE ==
(ARM_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_ARM64:
                        FOR (I = 0; I < INSN->DETAIL->ARM64.OP_COUNT; I++)
                                IF (INSN->DETAIL->ARM64.OPERANDS[I].TYPE ==
(ARM64_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_X86:
                        FOR (I = 0; I < INSN->DETAIL->X86.OP_COUNT; I++)
                                IF (INSN->DETAIL->X86.OPERANDS[I].TYPE ==
(X86_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_MIPS:
                        FOR (I = 0; I < INSN->DETAIL->MIPS.OP_COUNT; I++)
                                IF (INSN->DETAIL->MIPS.OPERANDS[I].TYPE ==
(MIPS_OP_TYPE)OP_TYPE)
                                        COUNT++;
```

```
                        BREAK;
                CASE CS_ARCH_PPC:
                        FOR (I = 0; I < INSN->DETAIL->PPC.OP_COUNT; I++)
                                IF (INSN->DETAIL->PPC.OPERANDS[I].TYPE ==
(PPC_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_SPARC:
                        FOR (I = 0; I < INSN->DETAIL->SPARC.OP_COUNT; I++)
                                IF (INSN->DETAIL->SPARC.OPERANDS[I].TYPE ==
(SPARC_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_SYSZ:
                        FOR (I = 0; I < INSN->DETAIL->SYSZ.OP_COUNT; I++)
                                IF (INSN->DETAIL->SYSZ.OPERANDS[I].TYPE ==
(SYSZ_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_XCORE:
                        FOR (I = 0; I < INSN->DETAIL->XCORE.OP_COUNT; I++)
                                IF (INSN->DETAIL->XCORE.OPERANDS[I].TYPE ==
(XCORE_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_M68K:
                        FOR (I = 0; I < INSN->DETAIL->M68K.OP_COUNT; I++)
                                IF (INSN->DETAIL->M68K.OPERANDS[I].TYPE ==
(M68K_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_TMS320C64X:
                        FOR (I = 0; I < INSN->DETAIL->TMS320C64X.OP_COUNT; I++)
                                IF (INSN->DETAIL->TMS320C64X.OPERANDS[I].TYPE ==
(TMS320C64X_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_M680X:
                        FOR (I = 0; I < INSN->DETAIL->M680X.OP_COUNT; I++)
                                IF (INSN->DETAIL->M680X.OPERANDS[I].TYPE ==
(M680X_OP_TYPE)OP_TYPE)
                                        COUNT++;
                        BREAK;
                CASE CS_ARCH_EVM:
#IF 0
                        FOR (I = 0; I < INSN->DETAIL->EVM.OP_COUNT; I++)
                                IF (INSN->DETAIL->EVM.OPERANDS[I].TYPE ==
(EVM_OP_TYPE)OP_TYPE)
                                        COUNT++;
#ENDIF
                        BREAK;
        }

        RETURN COUNT;
}
```

Example of x86 instruction opcode type (judgment register opcode)

Code:

```
typedef enum x86_op_type {
        X86_OP_INVALID = 0, ///< = CS_OP_INVALID (Uninitialized).
        X86_OP_REG, ///< = CS_OP_REG (Register opcode).
        X86_OP_IMM, ///< = CS_OP_IMM (Opcode now).
        X86_OP_MEM, ///< = CS_OP_MEM (Memory opcode).
} x86_op_type;
#include <iostream>
#include <stdio.h>

#include "capstone.h"
#include "platform.h"

using namespace std;

struct platform {
        cs_arch arch;
        cs_mode mode;
        unsigned char* code;
        size_t size;
        const char* comment;
        cs_opt_type opt_type;
        cs_opt_value opt_value;
};

static void print_string_hex(unsigned char* str, size_t len)
{
        unsigned char* c;

        printf("Code: ");
        for (c = str; c < str + len; c++) {
                printf("0x%02x ", *c & 0xff);
        }
        printf("\n");
}

static void test()
{

#define x86_code64
"\x55\x48\x8b\x05\xb8\x13\x00\x00\xe9\xea\xbe\xad\xde\xff\x25\x23\x01\x00\x00\xe8\xdf\xbe\xad
\xde\x74\xff"

        struct platform platforms[] = {
                {
                        cs_arch_x86,
                        cs_mode_64,
                        (unsigned char*)x86_code64,
                        sizeof(x86_code64) - 1,
                        "x86 64 (Intel syntax)"
```

```
                }, 
        };

        CSH HANDLE;
        UINT64_T ADDRESS;
        CS_INSN* INSN;
        CS_DETAIL* DETAIL;
        INT I;
        CS_ERR ERR;
        CONST UINT8_T* CODE;
        SIZE_T SIZE;

        FOR (I = 0; I < SIZEOF(PLATFORMS) / SIZEOF(PLATFORMS[0]); I++) {
                PRINTF("****************\N");
                PRINTF("PLATFORM: %S\N", PLATFORMS[I].COMMENT);
                ERR = CS_OPEN(PLATFORMS[I].ARCH, PLATFORMS[I].MODE, &HANDLE);
                IF (ERR) {
                        PRINTF("FAILED ON CS_OPEN() WITH ERROR RETURNED: %U\N", ERR);
                        ABORT();
                }

                IF (PLATFORMS[I].OPT_TYPE)
                        CS_OPTION(HANDLE, PLATFORMS[I].OPT_TYPE, PLATFORMS[I].OPT_VALUE);

                CS_OPTION(HANDLE, CS_OPT_DETAIL, CS_OPT_ON);

                INSN = CS_MALLOC(HANDLE);

                PRINT_STRING_HEX(PLATFORMS[I].CODE, PLATFORMS[I].SIZE);
                PRINTF("DISASM:\N");

                ADDRESS = 0X1000;
                CODE = PLATFORMS[I].CODE;
                SIZE = PLATFORMS[I].SIZE;
                WHILE (CS_DISASM_ITER(HANDLE, &CODE, &SIZE, &ADDRESS, INSN)) {
                        INT N;

                        PRINTF("0X%" PRIX64 ":\T%S\T\T%S         ",
                                INSN->ADDRESS, INSN->MNEMONIC, INSN->OP_STR);
                        COUT << "IS REG:   " << CS_OP_COUNT(HANDLE, INSN, X86_OP_REG) <<
ENDL;   // DETERMINE WHETHER IT IS A REGISTERED OPCODE
                        COUT << ENDL;

                PRINTF("\N");
                CS_FREE(INSN, 1);
                CS_CLOSE(&HANDLE);
        }
}

INT MAIN()
{
        TEST();

        RETURN 0;
```

```
}
```

output

```
Platform: X86 64 (Intel syntax)
Code: 0x55 0x48 0x8b 0x05 0xb8 0x13 0x00 0x00 0xe9 0xea 0xbe 0xad 0xde 0xff 0x25 0x23 0x01 0x00 0x00 0xe8 0xdf 0xbe 0xad
 0xde 0x74 0xff
Disasm:
0x1000: push            rbp          is REG:   1

0x1001: mov             rax, qword ptr [rip + 0x13b8]          is REG:   1

0x1008: jmp             0xffffffffdeadcef7          is REG:   0

0x100d: jmp             qword ptr [rip + 0x123]          is REG:   0

0x1013: call            0xffffffffdeadcef7          is REG:   0

0x1018: je              0x1019          is REG:   0
```

cs_op_index

```
int CAPSTONE_API cs_op_index(csh handle, const cs_insn *insn, unsigned int op_type,
unsigned int position);
```

Retrieve the operand of the given type in <arch>.The position in the operands[] array, use the returned position to access the operand

Note: This API is only available when the detail option is ON (OFF by default).

Handle: The handle returned by cs_open()

insn: Disassembly instruction structure received from cs_disasm() or cs_disasm_iter()

op_type: The type of operand to be found.

Position: The position of the operand to be found.The range must be within`[1, cs_op_data(handle, insn, op_type)]`

return:`<arch> of the instruction insn.operands[]'The index of the operand of the given type op_type in the array, returns -1 when it fails.

Code:

```
INT CAPSTONE_API CS_OP_INDEX(CSH UD, CONST CS_INSN *INSN, UNSIGNED INT OP_TYPE,
                    UNSIGNED INT POST)
{
          STRUCT CS_STRUCT *HANDLE;
          UNSIGNED INT COUNT = 0, I;
          IF (!UD)
                    RETURN -1;

          HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

          IF (!HANDLE->DETAIL) {
                    HANDLE->ERRNUM = CS_ERR_DETAIL;
                    RETURN -1;
          }

          IF (!INSN->ID) {
                    HANDLE->ERRNUM = CS_ERR_SKIPDATA;
                    RETURN -1;
          }

          IF (!INSN->DETAIL) {
                    HANDLE->ERRNUM = CS_ERR_DETAIL;
                    RETURN -1;
          }

          HANDLE->ERRNUM = CS_ERR_OK;

          SWITCH (HANDLE->ARCH) {
                    DEFAULT:
                              HANDLE->ERRNUM = CS_ERR_HANDLE;
                              RETURN -1;
                    CASE CS_ARCH_ARM:
                              FOR (I = 0; I < INSN->DETAIL->ARM.OP_COUNT; I++) {
                                        IF (INSN->DETAIL->ARM.OPERANDS[I].TYPE ==
(ARM_OP_TYPE)OP_TYPE)
                                                  COUNT++;
                                        IF (COUNT == POST)
                                                  RETURN I;
                              }
                              BREAK;
                    CASE CS_ARCH_ARM64:
                              FOR (I = 0; I < INSN->DETAIL->ARM64.OP_COUNT; I++) {
                                        IF (INSN->DETAIL->ARM64.OPERANDS[I].TYPE ==
(ARM64_OP_TYPE)OP_TYPE)
                                                  COUNT++;
                                        IF (COUNT == POST)
                                                  RETURN I;
                              }
                              BREAK;
                    CASE CS_ARCH_X86:
                              FOR (I = 0; I < INSN->DETAIL->X86.OP_COUNT; I++) {
                                        IF (INSN->DETAIL->X86.OPERANDS[I].TYPE ==
(X86_OP_TYPE)OP_TYPE)
```

```
                                        COUNT++;
                                IF (COUNT == POST)
                                        RETURN I;
                        }
                        BREAK;
                CASE CS_ARCH_MIPS:
                        FOR (I = 0; I < INSN->DETAIL->MIPS.OP_COUNT; I++) {
                                IF (INSN->DETAIL->MIPS.OPERANDS[I].TYPE ==
(MIPS_OP_TYPE)OP_TYPE)
                                        COUNT++;
                                IF (COUNT == POST)
                                        RETURN I;
                        }
                        BREAK;
                CASE CS_ARCH_PPC:
                        FOR (I = 0; I < INSN->DETAIL->PPC.OP_COUNT; I++) {
                                IF (INSN->DETAIL->PPC.OPERANDS[I].TYPE ==
(PPC_OP_TYPE)OP_TYPE)
                                        COUNT++;
                                IF (COUNT == POST)
                                        RETURN I;
                        }
                        BREAK;
                CASE CS_ARCH_SPARC:
                        FOR (I = 0; I < INSN->DETAIL->SPARC.OP_COUNT; I++) {
                                IF (INSN->DETAIL->SPARC.OPERANDS[I].TYPE ==
(SPARC_OP_TYPE)OP_TYPE)
                                        COUNT++;
                                IF (COUNT == POST)
                                        RETURN I;
                        }
                        BREAK;
                CASE CS_ARCH_SYSZ:
                        FOR (I = 0; I < INSN->DETAIL->SYSZ.OP_COUNT; I++) {
                                IF (INSN->DETAIL->SYSZ.OPERANDS[I].TYPE ==
(SYSZ_OP_TYPE)OP_TYPE)
                                        COUNT++;
                                IF (COUNT == POST)
                                        RETURN I;
                        }
                        BREAK;
                CASE CS_ARCH_XCORE:
                        FOR (I = 0; I < INSN->DETAIL->XCORE.OP_COUNT; I++) {
                                IF (INSN->DETAIL->XCORE.OPERANDS[I].TYPE ==
(XCORE_OP_TYPE)OP_TYPE)
                                        COUNT++;
                                IF (COUNT == POST)
                                        RETURN I;
                        }
                        BREAK;
                CASE CS_ARCH_M68K:
                        FOR (I = 0; I < INSN->DETAIL->M68K.OP_COUNT; I++) {
                                IF (INSN->DETAIL->M68K.OPERANDS[I].TYPE ==
(M68K_OP_TYPE)OP_TYPE)
                                        COUNT++;
```

```
                                        IF (COUNT == POST)
                                                RETURN I;
                                }
                                BREAK;
                        CASE CS_ARCH_TMS320C64X:
                                FOR (I = 0; I < INSN->DETAIL->TMS320C64X.OP_COUNT; I++) {
                                        IF (INSN->DETAIL->TMS320C64X.OPERANDS[I].TYPE ==
(TMS320C64X_OP_TYPE)OP_TYPE)
                                                COUNT++;
                                        IF (COUNT == POST)
                                                RETURN I;
                                }
                                BREAK;
                        CASE CS_ARCH_M680X:
                                FOR (I = 0; I < INSN->DETAIL->M680X.OP_COUNT; I++) {
                                        IF (INSN->DETAIL->M680X.OPERANDS[I].TYPE ==
(M680X_OP_TYPE)OP_TYPE)
                                                COUNT++;
                                        IF (COUNT == POST)
                                                RETURN I;
                                }
                                BREAK;
        }

        RETURN -1;
}
```

Exapmle with program

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

STRUCT PLATFORM {
        CS_ARCH ARCH;
        CS_MODE MODE;
        UNSIGNED CHAR* CODE;
        SIZE_T SIZE;
        CONST CHAR* COMMENT;
        CS_OPT_TYPE OPT_TYPE;
        CS_OPT_VALUE OPT_VALUE;
};

STATIC VOID PRINT_STRING_HEX(UNSIGNED CHAR* STR, SIZE_T LEN)
{
        UNSIGNED CHAR* C;

        PRINTF("CODE: ");
        FOR (C = STR; C < STR + LEN; C++) {
                PRINTF("0X%02X ", *C & 0XFF);
```

```
        }
        printf("\n");
}

static void test()
{
#define X86_CODE64
"\x55\x48\x8b\x05\xb8\x13\x00\x00\xe9\xea\xbe\xad\xde\xff\x25\x23\x01\x00\x00\xe8\xdf\xbe\xad
\xde\x74\xff"
        struct platform platforms[] = {
                {
                        CS_ARCH_X86,
                        CS_MODE_64,
                        (unsigned char*)X86_CODE64,
                        sizeof(X86_CODE64) - 1,
                        "X86 64 (Intel syntax)"
                },
        };

        csh handle;
        uint64_t address;
        cs_insn* insn;
        cs_detail* detail;
        int i;
        cs_err err;
        const uint8_t* code;
        size_t size;

        cs_x86* x86;

        int count;

        for (i = 0; i < sizeof(platforms) / sizeof(platforms[0]); i++) {
                printf("****************\n");
                printf("Platform: %s\n", platforms[i].comment);
                err = cs_open(platforms[i].arch, platforms[i].mode, &handle);
                if (err) {
                        printf("Failed on cs_open() with error returned: %u\n", err);
                        abort();
                }

                if (platforms[i].opt_type)
                        cs_option(handle, platforms[i].opt_type, platforms[i].opt_value);

                cs_option(handle, CS_OPT_DETAIL, CS_OPT_ON);

                insn = cs_malloc(handle);
                x86 = &(insn->detail->x86);
                print_string_hex(platforms[i].code, platforms[i].size);
                printf("Disasm:\n");

                address = 0x1000;
                code = platforms[i].code;
                size = platforms[i].size;
                while (cs_disasm_iter(handle, &code, &size, &address, insn)) {
```

```
                        INT N;

                        PRINTF("0X%" PRIX64 ":\T%S\T\T%S         ",
                                INSN->ADDRESS, INSN->MNEMONIC, INSN->OP_STR);
                        COUT << ENDL;

                        COUNT = CS_OP_COUNT(HANDLE, INSN, X86_OP_IMM);  // FIND IMMEDIATE
NUMBER
                        IF (COUNT) {
                                PRINTF("\TIMM_COUNT: %U\N", COUNT);
                                FOR (I = 1; I < COUNT + 1; I++) {
                                        INT INDEX = CS_OP_INDEX(HANDLE, INSN, X86_OP_IMM,
I);
                                        PRINTF("\TIMMS[%U]: 0X%" PRIX64 "\N", I, X86-
>OPERANDS[INDEX].IMM);
                                        IF (X86->ENCODING.IMM_OFFSET != 0) {
                                                PRINTF("\TIMM_OFFSET: 0X%X\N", X86-
>ENCODING.IMM_OFFSET);
                                        }
                                        IF (X86->ENCODING.IMM_SIZE != 0) {
                                                PRINTF("\TIMM_SIZE: 0X%X\N", X86-
>ENCODING.IMM_SIZE);
                                        }
                                }
                        }
                }

                PRINTF("\N");
                CS_FREE(INSN, 1);
                CS_CLOSE(&HANDLE);
        }
}

INT MAIN()
{
        TEST();
        RETURN 0;
}
```

output

```
Platform: X86 64 (Intel syntax)
Code: 0x55 0x48 0x8b 0x05 0xb8 0x13 0x00 0x00 0xe9 0xea 0xbe 0xad 0xde 0xff 0x25 0x23 0x01 0x00 0x00 0xe8 0xdf 0xbe 0xad
 0xde 0x74 0xff
Disasm:
0x1000: push            rbp
0x1001: mov             rax, qword ptr [rip + 0x13b8]
0x1008: jmp             0xffffffffdeadcef7
        imm_count: 1
        imms[1]: 0xffffffffdeadcef7
        imm_offset: 0x1
        imm_size: 0x4
0x100d: jmp             qword ptr [rip + 0x123]
0x1013: call            0xffffffffdeadcef7
        imm_count: 1
        imms[1]: 0xffffffffdeadcef7
        imm_offset: 0x1
        imm_size: 0x4
0x1018: je              0x1019
        imm_count: 1
        imms[1]: 0x1019
        imm_offset: 0x1
        imm_size: 0x1
```

cs_regs_access

```
cs_err CAPSTONE_API cs_regs_access(csh handle, const cs_insn *insn,
                 cs_regs regs_read, uint8_t *regs_read_count,
                 cs_regs regs_write, uint8_t *regs_write_count);
```

Retrieve all registers accessed explicitly or implicitly by an instruction
Note: In ”diet" mode, this API is not available because the engine does not store registers

Handle: The handle returned by cs_open()

insn: Disassembly instruction structure returned from cs_disasm() or cs_disasm_iter()

regs_read: When returned, this array contains all the registers read by the instruction.

regs_read_data: The number of registers stored in the regs_read array.

regs_write: When returned, this array contains all the registers modified by the instruction.

regs_write_data: The number of registers stored in the regs_write array.

CS_ERR_OK is returned on success, and other values are returned on failure (please refer to cs_err enum for detailed errors).

Code:

```
CS_ERR CAPSTONE_API CS_REGS_ACCESS(CSH UD, CONST CS_INSN *INSN,
                    CS_REGS REGS_READ, UINT8_T *REGS_READ_COUNT,
                    CS_REGS REGS_WRITE, UINT8_T *REGS_WRITE_COUNT)
{
          STRUCT CS_STRUCT *HANDLE;

          IF (!UD)
                    RETURN -1;

          HANDLE = (STRUCT CS_STRUCT *)(UINTPTR_T)UD;

#IFDEF CAPSTONE_DIET
          // THIS API DOES NOT WORK IN DIET MODE
          HANDLE->ERRNUM = CS_ERR_DIET;
          RETURN CS_ERR_DIET;
#ELSE
          IF (!HANDLE->DETAIL) {
                    HANDLE->ERRNUM = CS_ERR_DETAIL;
                    RETURN CS_ERR_DETAIL;
          }

          IF (!INSN->ID) {
                    HANDLE->ERRNUM = CS_ERR_SKIPDATA;
                    RETURN CS_ERR_SKIPDATA;
          }

          IF (!INSN->DETAIL) {
                    HANDLE->ERRNUM = CS_ERR_DETAIL;
                    RETURN CS_ERR_DETAIL;
          }

          IF (HANDLE->REG_ACCESS) {
                    HANDLE->REG_ACCESS(INSN, REGS_READ, REGS_READ_COUNT, REGS_WRITE,
REGS_WRITE_COUNT);
          } ELSE {
                    // THIS ARCH IS UNSUPPORTED YET
                    HANDLE->ERRNUM = CS_ERR_ARCH;
                    RETURN CS_ERR_ARCH;
          }

          RETURN CS_ERR_OK;
#ENDIF
}
```

Example with program

Code:

```
#INCLUDE <IOSTREAM>
#INCLUDE <STDIO.H>

#INCLUDE "CAPSTONE.H"
#INCLUDE "PLATFORM.H"

USING NAMESPACE STD;

STRUCT PLATFORM {
          CS_ARCH ARCH;
          CS_MODE MODE;
          UNSIGNED CHAR* CODE;
          SIZE_T SIZE;
          CONST CHAR* COMMENT;
          CS_OPT_TYPE OPT_TYPE;
          CS_OPT_VALUE OPT_VALUE;
};

STATIC VOID PRINT_STRING_HEX(UNSIGNED CHAR* STR, SIZE_T LEN)
{
          UNSIGNED CHAR* C;

          PRINTF("CODE: ");
          FOR (C = STR; C < STR + LEN; C++) {
                    PRINTF("0X%02X ", *C & 0XFF);
          }
          PRINTF("\N");
}

STATIC VOID TEST()
{
#DEFINE X86_CODE64
"\X55\X48\X8B\X05\XB8\X13\X00\X00\XE9\XEA\XBE\XAD\XDE\XFF\X25\X23\X01\X00\X00\XE8\XDF\XBE\XAD
\XDE\X74\XFF"
          STRUCT PLATFORM PLATFORMS[] = {
                    {
                              CS_ARCH_X86,
                              CS_MODE_64,
                              (UNSIGNED CHAR*)X86_CODE64,
                              SIZEOF(X86_CODE64) - 1,
                              "X86 64 (INTEL SYNTAX)"
                    },
          };

          CSH HANDLE;
          UINT64_T ADDRESS;
          CS_INSN* INSN;
          CS_DETAIL* DETAIL;
          INT I;
          CS_ERR ERR;
          CONST UINT8_T* CODE;
          SIZE_T SIZE;

          CS_X86* X86;
          CS_REGS REGS_READ, REGS_WRITE;
```

```
        UINT8_T REGS_READ_COUNT, REGS_WRITE_COUNT;

        INT COUNT;

        FOR (I = 0; I < SIZEOF(PLATFORMS) / SIZEOF(PLATFORMS[0]); I++) {
                PRINTF("****************\N");
                PRINTF("PLATFORM: %S\N", PLATFORMS[I].COMMENT);
                ERR = CS_OPEN(PLATFORMS[I].ARCH, PLATFORMS[I].MODE, &HANDLE);
                IF (ERR) {
                        PRINTF("FAILED ON CS_OPEN() WITH ERROR RETURNED: %U\N", ERR);
                        ABORT();
                }

                IF (PLATFORMS[I].OPT_TYPE)
                        CS_OPTION(HANDLE, PLATFORMS[I].OPT_TYPE, PLATFORMS[I].OPT_VALUE);

                CS_OPTION(HANDLE, CS_OPT_DETAIL, CS_OPT_ON);

                INSN = CS_MALLOC(HANDLE);
                X86 = &(INSN->DETAIL->X86);
                PRINT_STRING_HEX(PLATFORMS[I].CODE, PLATFORMS[I].SIZE);
                PRINTF("DISASM:\N");

                ADDRESS = 0X1000;
                CODE = PLATFORMS[I].CODE;
                SIZE = PLATFORMS[I].SIZE;
                WHILE (CS_DISASM_ITER(HANDLE, &CODE, &SIZE, &ADDRESS, INSN)) {
                        INT N;

                        PRINTF("0X%" PRIX64 ":\T%S\T\T%S         ",
                                INSN->ADDRESS, INSN->MNEMONIC, INSN->OP_STR);
                        COUT << ENDL;

                        IF (!CS_REGS_ACCESS(HANDLE, INSN,       // ALL READ AND MODIFIED
REGISTERS FOR EACH INSTRUCTION
                                REGS_READ, &REGS_READ_COUNT,
                                REGS_WRITE, &REGS_WRITE_COUNT)) {
                                IF (REGS_READ_COUNT) {
                                        PRINTF("\TREGISTERS READ:");
                                        FOR (I = 0; I < REGS_READ_COUNT; I++) {
                                                PRINTF(" %S", CS_REG_NAME(HANDLE,
REGS_READ[I]));
                                        }
                                        PRINTF("\N");
                                }

                                IF (REGS_WRITE_COUNT) {
                                        PRINTF("\TREGISTERS MODIFIED:");
                                        FOR (I = 0; I < REGS_WRITE_COUNT; I++) {
                                                PRINTF(" %S", CS_REG_NAME(HANDLE,
REGS_WRITE[I]));
                                        }
                                        PRINTF("\N");
                                }
```

```
                        }
                }

                PRINTF("\N");
                CS_FREE(INSN, 1);
                CS_CLOSE(&HANDLE);
        }
}

INT MAIN()
{
        TEST();
        RETURN 0;
}
```

output

```
****************
Platform: X86 64 (Intel syntax)
Code: 0x55 0x48 0x8b 0x05 0xb8 0x13 0x00 0x00 0xe9 0xea 0xbe 0xad 0xde 0xff 0x25 0x23 0x01 0x00 0x00 0xe8 0xdf 0xbe 0xad
 0xde 0x74 0xff
Disasm:
0x1000: push            rbp
        Registers read: rsp rbp
        Registers modified: rsp
0x1001: mov             rax, qword ptr [rip + 0x13b8]
        Registers read: rip
        Registers modified: rax
0x1008: jmp             0xffffffffdeadcef7
0x100d: jmp             qword ptr [rip + 0x123]
        Registers read: rip
0x1013: call            0xffffffffdeadcef7
        Registers read: rsp rip
        Registers modified: rsp
0x1018: je              0x1019
        Registers read: rflags
```