

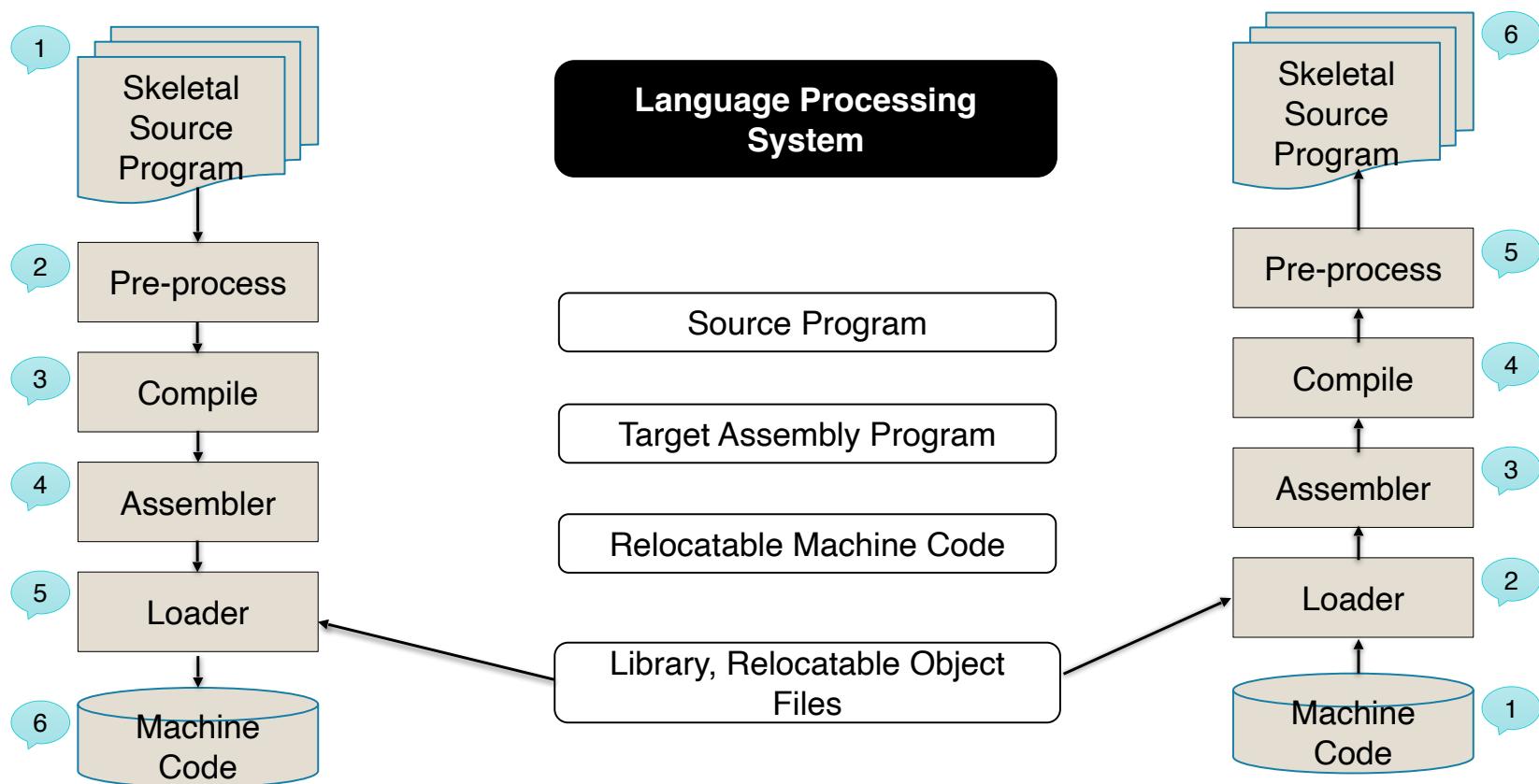
# **EECS 388: Embedded Systems**

A practical session on how to do  
binary reverse engineering

## OVERVIEW

- What is Bitstream/Binary/Software Reverse Engineering(RE)?
- Motivation
- Systematic way of doing RE
- Our goal
- Intro to 8051 ISA
- Intro to KEIL
- Example
- Assignment :')

# INTRODUCTION



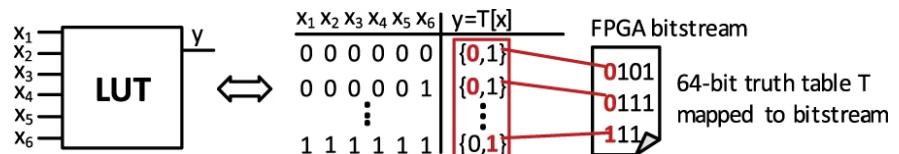
## BITSTREAM/BINARY/SOFTWARE REV. ENG.

- Understand the properties of machine code/object code.
- Each architecture has fixed set of opcode, operand and register naming.
- Can be Static or Dynamic.
- In digital electronics, a bitstream is similar to the sequence of binary digits but implies configuration of a programmable device or ASIC.

Machine	Binary	Assembly
E8	1110 1000	MOV A, R0
EF	1110 1111	MOV A, R7
F8	1111 1000	MOV R0, A
FF	1111 1111	MOV R7, A
	1111 1XXX	MOV RXXX, A

Code	Binary	Assembly
74 25	0111 0100 0010 0101	MOV A, #25H
78 25	0111 1000 0010 0101	MOV R0, #25H
7F 25	0111 1111 0010 0101	MOV R7, #25H

[MicroController 4 All: Session 5 - 8051 Addressing Modes](#)



[Bitstream Fault Injections \(BiFI\)—Automated Fault Attacks Against SRAM-Based FPGAs \(computer.org\)](#)

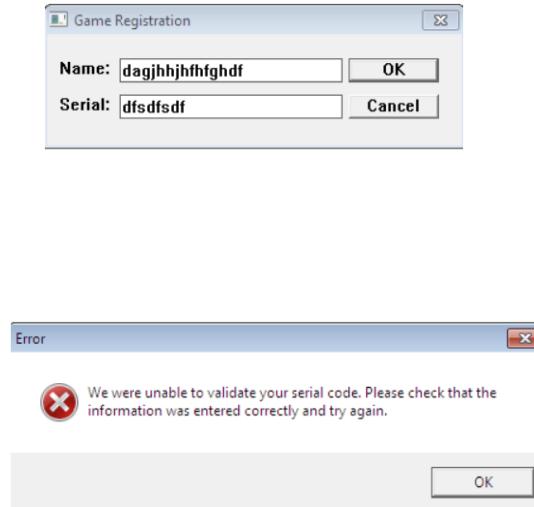
# WHY REV. ENG. ?

- Legitimate purposes
  - Understand the program
  - Identify vulnerabilities/weakness
  - Analyze security vulnerabilities
- Malicious purposes
  - Unauthorized overproduction / IP piracy
  - Develop malware / create malicious version
  - Trojan insertion

The collage includes:

- A screenshot of the Threatpost website header.
- A screenshot of the BleepingComputer Insider page featuring a story about the WannaCry ransomware attack.
- A screenshot of a Wikipedia article on the WannaCry ransomware attack.
- A large screenshot of a wanted poster for a person involved in a wire fraud conspiracy, with the text "WANTED BY THE FBI" and "Conspiracy to Commit Wire Fraud; Conspiracy to Commit Computer-Related Fraud (Computer Intrusion)".
- A small screenshot of a ransom note from the WannaCry attack.

# EXAMPLE



Step  
1

```
.text:00401157 mov    eax, [esp+318h+var_30C]
.text:00401158 cmp    eax, 5
.text:0040115E jb     short loc_401189 ; uType
.text:00401160 push   10h
.text:00401162 push   offset Caption ; "Error"
.text:00401164 push   offset Text   ; "Please enter a longer name."
.text:00401166 push   edi
.text:00401168 call   ds:MessageBoxA ; hWnd
.text:00401173 xor    eax, eax
```

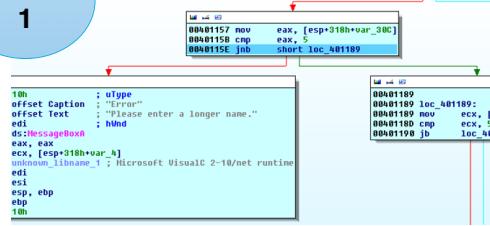
IDA Pro interface showing assembly code and memory dump. The assembly code pane shows the same instructions as above, with the line ".text:0040115E jb short loc\_401189 ; uType" highlighted in red. The memory dump pane shows the raw bytes of the assembly code, with the same line highlighted in blue. A third pane shows the assembly code again, with the same line highlighted in green.

Tool used: IDA Pro

# EXAMPLE

```
.text:00401157    mov    eax, [esp+318h+var_30C]
.text:0040115B    ret
.text:0040115E    dbh    short loc_401189 ; uType
.text:00401160    push   10h
.text:00401162    push   offset Caption ; "Error"
.text:00401167    push   offset Text ; "Please enter a longer name."
.text:0040116C    push   edi
.text:0040116D    call   ds:MessageBoxBox
.text:00401173    xor    eax, eax
```

**Step 1**



**OllyDbg - victim.exe**

File View Debug Plugins Options Window Help

C CPU - main thread, module victim

```
0040115B: 59      POP ECX
0040115C: C2      RETN
0040115D: 60      PUSH 60
0040115E: E8 7F830000 CALL victim.00401170
0040115F: BF 94000000 MOV EDI,94
00401160: 8B77 00000000 MOV EBX,EDI
00401161: 8B7F 10000000 CALL kernel32.00402690
00401162: 8965 E8    MOV QWORD PTR SS:[EBP-18],ESP
00401163: BFB4      MOV ESI,ESP
00401164: 893E      MOV QWORD PTR DS:[ESI],EDI
00401165: 56      PUSH ESI
00401166: C744000000 MOV ECX,QWORD PTR DS:[<&KERNEL32.GetVersion>]
00401167: 890D BC32400000 MOV EDX,QWORD PTR DS:[4082280],ECX
00401168: 8B46 04    MOV EBX,DWORD PTR DS:[ESI+4]
00401169: A3 C88240000 MOV QWORD PTR DS:[4082280],EBX
0040116A: B856 08    MOV EDX,DWORD PTR DS:[ESI+8]
0040116B: 8915 CC8240000 MOV QWORD PTR DS:[40822C0],EDX
0040116C: 8904 08    MOV ECX,DWORD PTR DS:[ESI+C]
0040116D: 81E5 FF7F0000 AND ES1,7FF
0040116E: 8935 C08240000 MOV QWORD PTR DS:[40822C0],ESI
0040116F: 83F9 02    CMP ECX,2
00401170: 74 0C    JE SHORT victim.00401640
00401171: 81CE 00000000 MOV ES1,8000
00401172: 8904 08    MOV ECX,DWORD PTR DS:[4082200],ESI
00401173: C1E9 08    SHL ECX,8
00401174: 803C ADD ECX,EDX
```

**Step 2**

**Entry Point**

GetVersionInformation

GetVersionExA

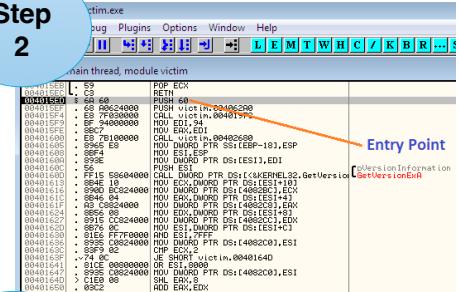
**Tool used: OllyDbg**

**Step 3**

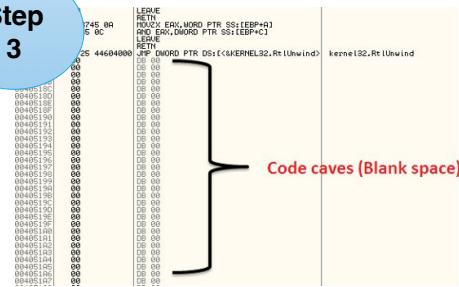
**Code caves (Blank space)**

# EXAMPLE

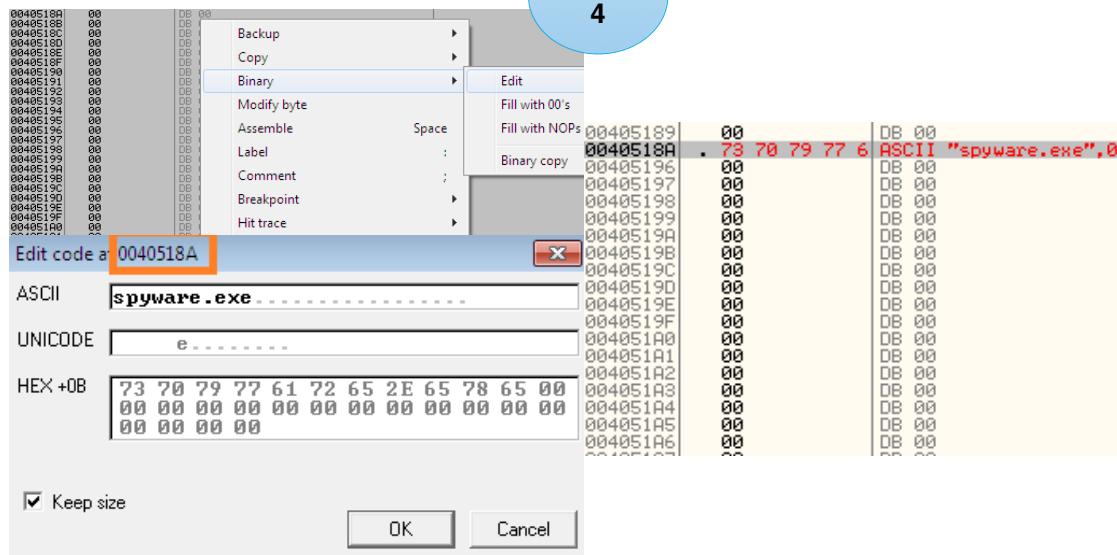
Step  
2



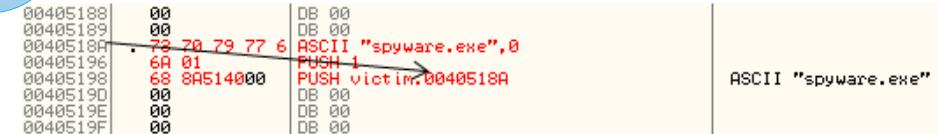
Step  
3



Step  
4



Step  
5



# EXAMPLE

Step  
4

```
00405189 00 00 DB 00
0040518A 73 70 79 77 6 ASCII "spyware.exe",0
00405196 00 00 DB 00
00405197 00 00 DB 00
00405198 00 00 DB 00
00405199 00 00 DB 00
0040519A 00 00 DB 00
0040519B 00 00 DB 00
0040519C 00 00 DB 00
0040519D 00 00 DB 00
0040519E 00 00 DB 00
0040519F 00 00 DB 00
004051A0 00 00 DB 00
004051A1 00 00 DB 00
004051A2 00 00 DB 00
004051A3 00 00 DB 00
004051A4 00 00 DB 00
004051A5 00 00 DB 00
004051A6 00 00 DB 00
```

Step  
5

```
00405189 00 00 DB 00
0040518A 00 00 DB 00
0040518B 73 70 79 77 6 ASCII "spyware.exe",0
0040518C 6A 01 PUSH 1
00405196 00 00 DB 00
00405197 68 8A514000 PUSH victim.0040518A
00405198 00 00 DB 00
00405199 00 00 DB 00
0040519A 00 00 DB 00
0040519B 00 00 DB 00
0040519C 00 00 DB 00
0040519D 00 00 DB 00
0040519E 00 00 DB 00
0040519F 00 00 DB 00
```

00405189 00 00 DB 00
0040518A 00 00 DB 00
0040518B 73 70 79 77 6 ASCII "spyware.exe",0
0040518C 6A 01 PUSH 1
0040518D 68 8A514000 PUSH victim.0040518A
0040518E E8 DB94C875 CALL kernel32.WinExec
0040518F 00 00 DB 00
00405190 00 00 DB 00

0040115E . 73 29 JNB SHORT victim.00401189
00401160 . 6A 10 PUSH 10
00401162 . 68 00624000 PUSH victim.00406200
00401164 . 68 E4614000 PUSH victim.004061E4
00401166 . 57 PUSH EDI
00401168 . FF15 00614000 CALL DWORD PTR DS:[<&USER32.MessageBoxA]
00401170 . 33C0 XOR EAX,EAX
00401172 . BB8C24 140300 MOV ECX,DWORD PTR SS:[ESP+314]
00401174 . E8 39040000 CALL victim.004015BA

Style = MB\_OK|MB\_ICONHAND|MB\_APPLMODAL  
Title = "Error"  
Text = "Please enter a longer name."  
hOwner = MessageBoxA

Step  
7  
= 1

0040518A 73 70 79 77 6 ASCII "spyware.exe",0
**0040518B 6A 01 PUSH 1**
0040518C 68 8A514000 PUSH victim.0040518A
0040518D E8 DB94C875 CALL kernel32.WinExec
0040518E ^0F83 E1BFFFFFF JNB victim.00401189

Step  
8

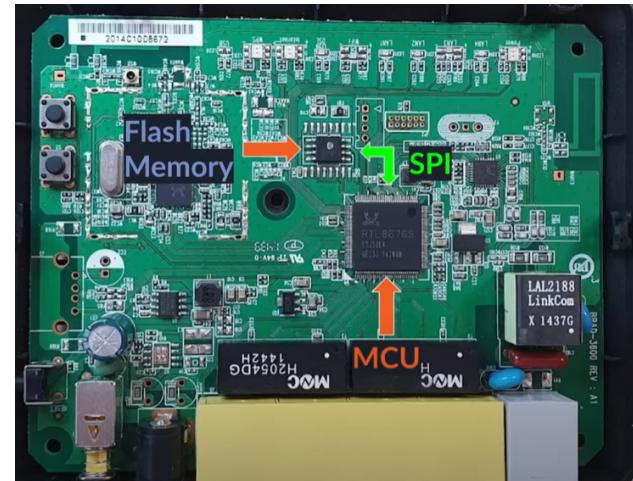
(This was JNB SHORT 00401189 earlier before alter)

0040115B 1. 83F8 05 CMP EAX,5
**0040115C 0F83 32400000 JNB victim.00405196**
00401164 90 NOP
00401165 90 NOP
00401166 90 NOP
00401168 . 68 E4614000 PUSH victim.004061E4
0040116C . 57 PUSH EDI
0040116D . FF15 00614000 CALL DWORD PTR DS:[<&USER32.MessageBoxA]
00401170 . 33C0 XOR EAX,EAX
00401172 . BB8C24 140300 MOV ECX,DWORD PTR SS:[ESP+314]
00401174 . E8 39040000 CALL victim.004015BA

Source : Injecting spyware in an EXE (code injection) |Infosec Resources  
(infosecinstitute.com)

## REV. ENG. IN EMBEDDED DEVICES

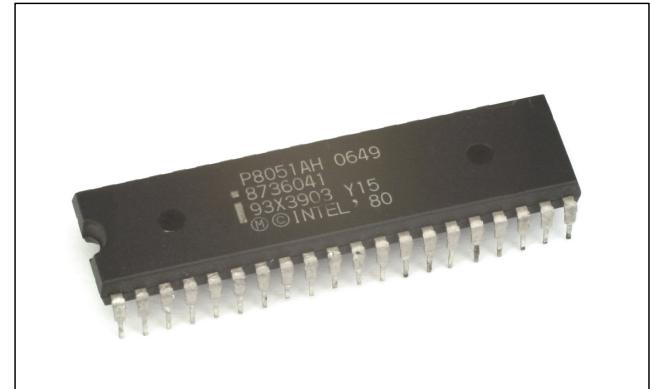
- Loading a program onto an embedded device required following steps:
  - Writing the program in behavior level i.e. C or C++
  - Compile the code into a machine readable format
  - Load the binary/executable image in device memory
- Can an attacker with physical access to the device steal the binary image?
- Yes! It is possible to extract the binary from its memory using tools such as JTAG debugger, logic analyzer, or firmware extraction tool.



Note: Using SPI (Serial Peripheral Interface) bus, MCU communicate with flash memory.

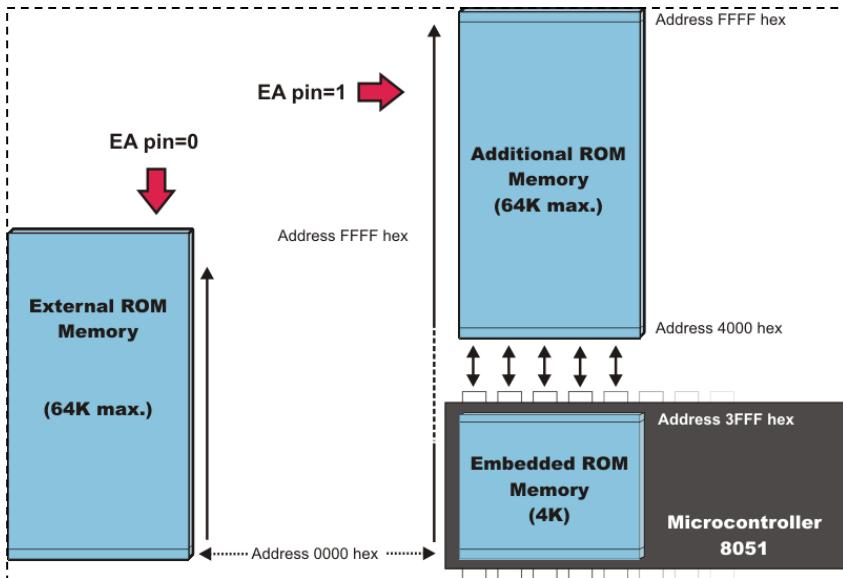
## INTRO TO 8051 MCU:

- 8-bit architecture introduced by intel in 1980s.
- 8-bit CPU, 4 KB of ROM, 128 bytes of RAM and 32 I/O pins
- Simple instruction set and can execute one instruction per cycle
- Can be programmed in assembly language or high-level languages such as C
- Still used today for embedded applications that require low power consumption, small size and most important for its simplicity.



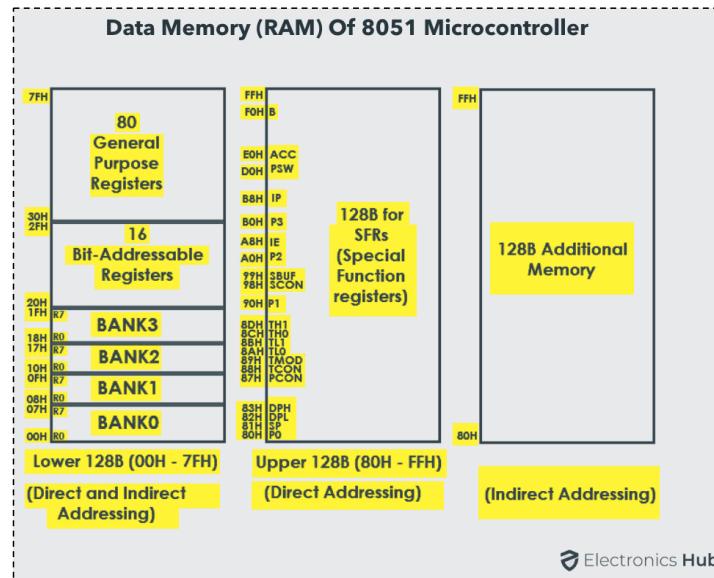
Source: [Intel 8051 - Wikipedia](#)

# MEMORY ORGANIZATION



## Program memory(ROM) :

1. Code being executed
2. Read only
3. Can store constant variables
4. External Access(EA) is a control pin



## Internal data memory(RAM):

1. Up to 256 bytes & stack starts from Bank 1
2. Three memory type specifiers can be used to refer to the internal data memory: data, idata, and bdata.

Source: 8051 Memory Organization (mikroe.com)

## INTRO TO 8051 MCU ISA:

OPCODE	USAGES	Details
MOV	Data Memory $\leftrightarrow$ Registers	Any data from internal 256 bytes of RAM
MOVC	Code Memory $\leftrightarrow$ Registers	Used for constants or an array of data
MOVX	Registers $\leftrightarrow$ External Memory	Same as MOV, but instead of internal data memory, it is used for external data memory
ADD	ADD op1, op2 ADD A, R1 $\leftrightarrow$ result in A(acc) ADD R1, R2 $\leftrightarrow$ <b>ERROR!</b>	One can be immediate, and the other must be some memory locations(register, accumulator(A), data memory).
SUBB	SUBB A, R1 $\leftrightarrow$ A - R1 result in A(acc) SUBB R1, R2 $\leftrightarrow$ <b>ERROR!</b>	Subtract with borrow. If A > R1 = no issues! If A < R1 = Borrow stores automatically!
DIV	DIV A, B $\leftrightarrow$ CORRECT! DIV A, R1 $\leftrightarrow$ <b>ERROR!</b>	A & B register can only be used! Result stored in A
MUL	MUL A, B $\leftrightarrow$ CORRECT! MUL A, R1 $\leftrightarrow$ <b>ERROR!</b>	A & B register can only be used!

## INTRO TO 8051 MCU ISA:

OPCODE	USAGES	Details
CJNE	CJNE OP1, OP2, LOC OP1 != OP2, LOC $\Leftrightarrow$ JUMP LOC OP1 == OP2 $\Leftrightarrow$ NOT JUMP	Compare Jump if Not Equal OP1 = A, R0-R7, @R0 - @R7 OP2 = immediate(direct value followed by #) LOC = Label, 1000H, 5000H etc.. SUBB makes a comparison between OP1 & OP2. If OP1 < OP2 ? C =1 : C = 0
DJNZ	DJNZ OP1, LOC OP1 != 0 $\Leftrightarrow$ JUMP LOC and OP1-- OP1 == 0 $\Leftrightarrow$ NOT JUMP	Decrement & JUMP on Non-Zero For i = 10; i <= 0; i -- OP1 = A, R0-R7, @R0 - @R7 LOC = Label, 1000H, 5000H etc..
JZ	JZ, LOC (relative address)	A is 0? JUMP to LOC : NOT JUMP
JNZ	JNZ, LOC (relative address)	A is not 0? JUMP to LOC : NOT JUMP
JC	JC, LOC (relative address)	If carry bit(C) is SET? JUMP to LOC : NOT JUMP C is located at PSW (Program Status Word) register.
JNC	JNC, LOC (relative address)	If carry bit is not SET ? JUMP to LOC : NOT JUMP
AJMP	AJMP, LOC (relative address)	Absolute Jump, Unconditional, No range
LJMP	LJMP, LOC (relative address)	Long Jump, Unconditional
SJMP	SJMP, LOC (relative address)	Short Jump, Unconditional, -127 to 127

## 8051 MCU ADDRESSING MODE:

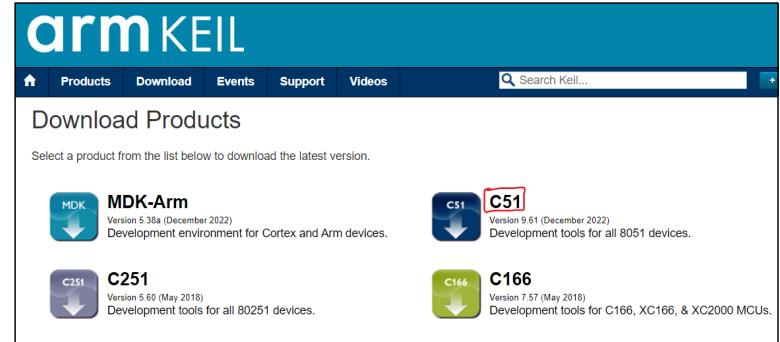
OPCODE Destination, Source	Source = {Data, Reg, Memory Location}
<b>Immediate Addressing Mode</b> ⇒ MOV A, #5h # is followed by hex data	Here, Source = Data Note: ⇒ MOV A, #Fh ⇔ <b>ERROR!</b> , ⇒ MOV A, #0Fh ⇔ CORRECT!
<b>Register Addressing Mode</b> ⇒ MOV A, R1 ⇒ MOV R2, R1 ⇔ ERROR!	Here, Source = Register 8051 is an ACCUMULATOR-based architecture.
<b>Direct or Absolute Addressing</b> What if the # is missing? ⇒ MOV A, 5h	Here, Source = Memory Location This is identified as the address. It will go to the address for 5h and pick up the data from the address.
<b>Indirect Addressing</b> ⇒ MOV R0, #20h ⇒ MOV A, @R0 ⇒ INC R0 # Now R0 = #21h	Here, Source = Array of data in memory, starting with "@" R0 and R1 registers are only involved in indirect addressing ⇒ "@R0" has the address where the data is located
<b>Index Addressing</b> ⇒ mov dptr, LOC ⇒ clr a ⇒ movc a, @a+dptr	Here, Source = 16-bit address DPTR ⇔ Data fetched from code memory DPTR ⇔ Data fetched from external RAM or data memory

## 8051 REGISTERS:

CPU Registers	Details
<b>Program Counter</b>	Keeps track of where the processor is in its execution of a program.
<b>Accumulator</b>	Special purpose register. The accumulator is typically a processor register that is optimized for fast access and is used in conjunction with arithmetic and logic operations.
<b>Stack Pointer</b>	Holds the memory address of the top of the stack
<b>Instruction Register</b>	Holds the current instruction that the processor is executing
<b>Address Register</b>	Holds the address of the next instruction to be executed
<b>Index Register</b>	Holds a memory address offset or an index value for data access operations. The index register is typically used to access data structures, such as arrays or tables, efficiently.

## PROGRAMMING IN KEIL:

- KEIL C51 is a cross-compiler for the 8051 microcontroller family.
- It supports a wide range of 8051 variants and extensions, such as code banking, memory models, and bit-addressable data types.
- IDE with a project manager, editor, debugger, and simulator.
- It includes a library of standard and vendor-specific header files, startup code, and runtime routines.
- It generates compact and efficient code that can be optimized for speed or size using various directives and options.



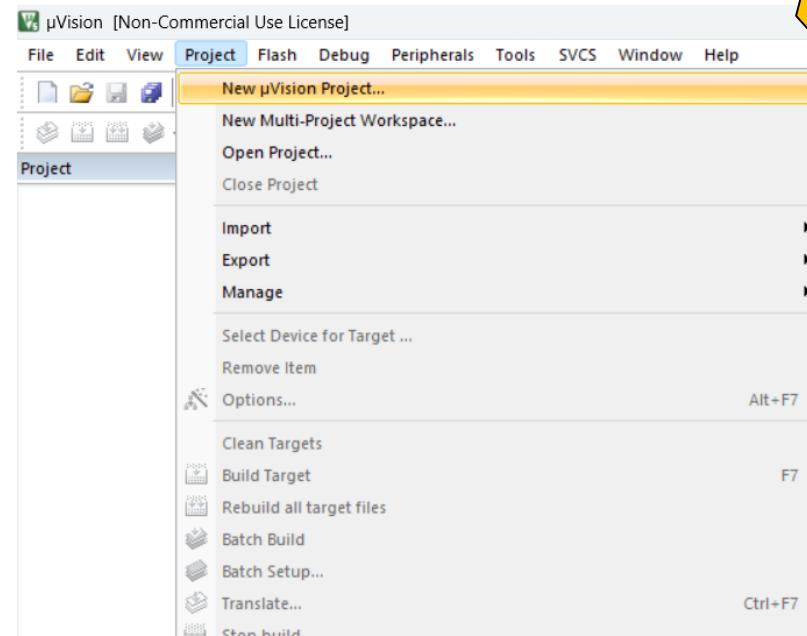
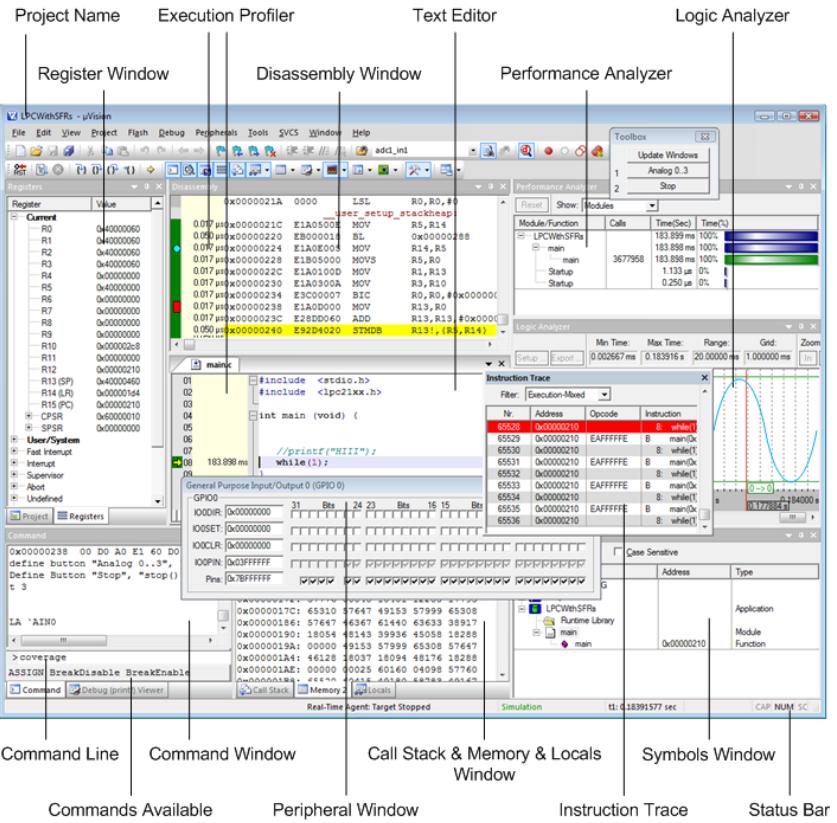
Download Link: [Keil Product Downloads](#)

How to Install: <https://youtu.be/MG595VN4r70>

**Note:** Run as administrator after installing the tool.

Keil doesn't support on MAC :(  
Statement: [GENERAL: Keil MDK 5 on a Mac \(arm.com\)](#)  
Solution: Virtual box / Cycle server

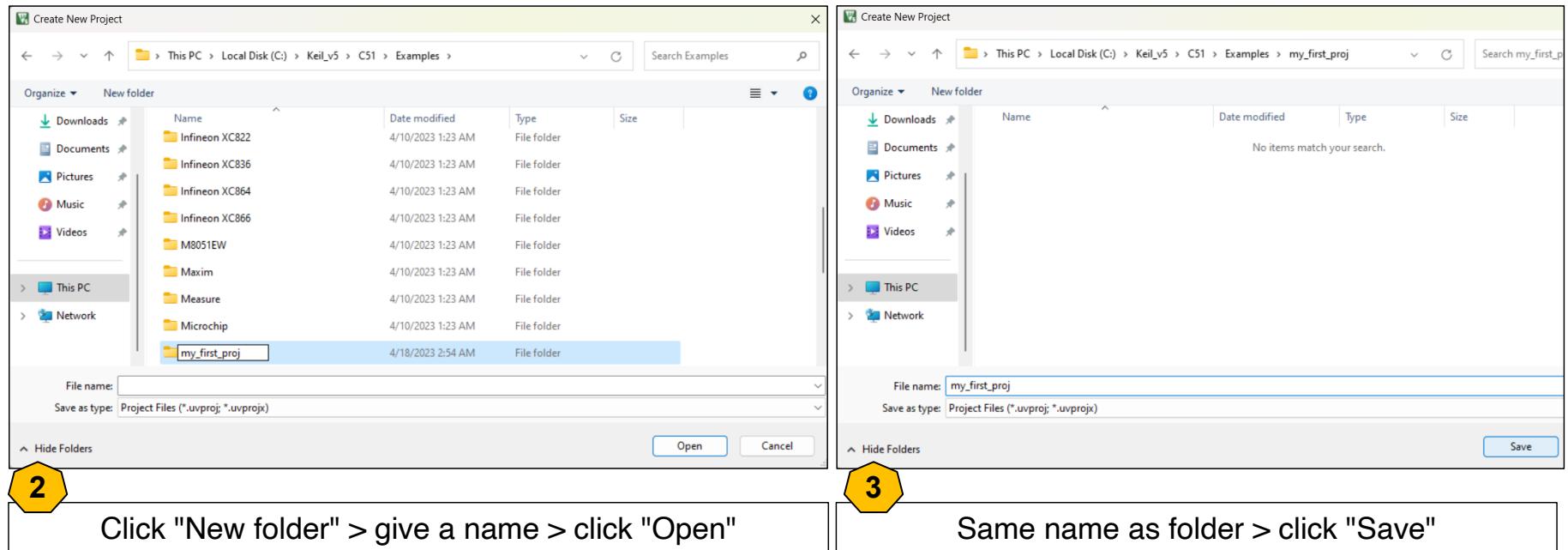
# KEIL TUTORIAL



Project > New uVision Project

1

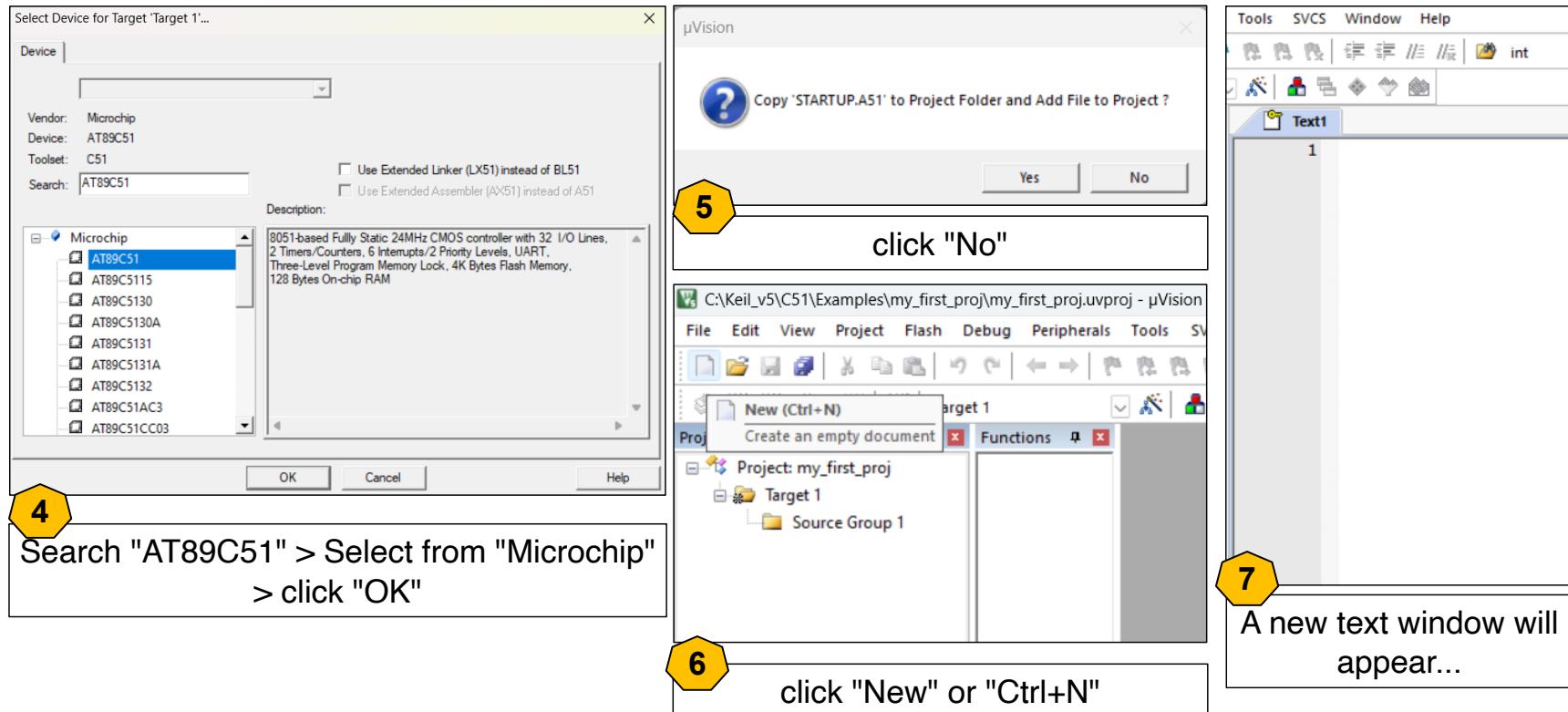
# KEIL TUTORIAL



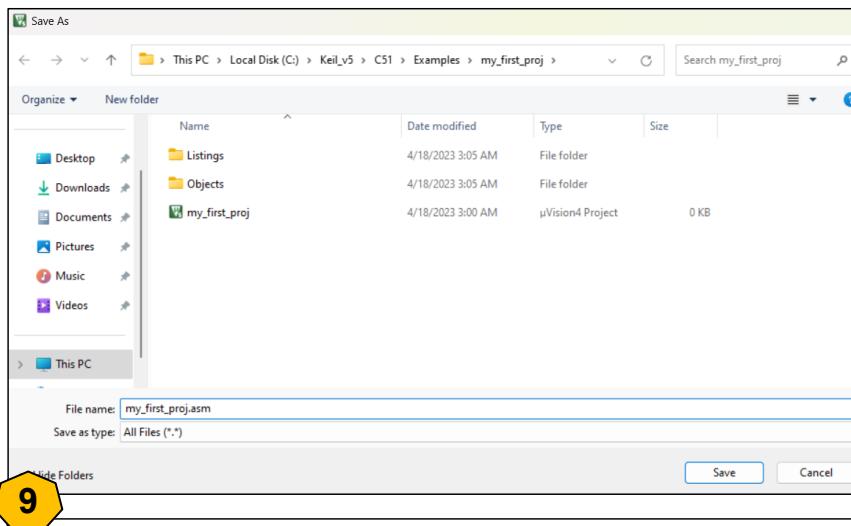
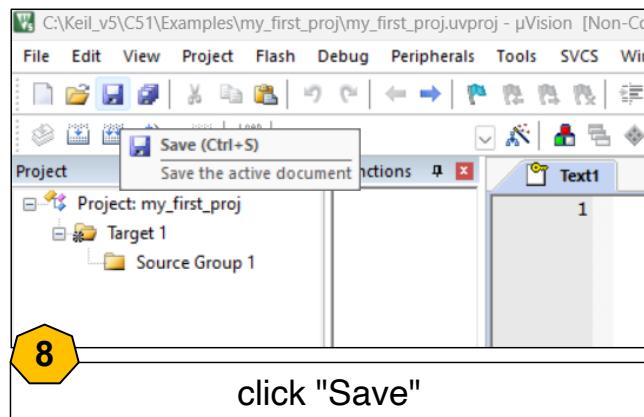
Click "New folder" > give a name > click "Open"

Same name as folder > click "Save"

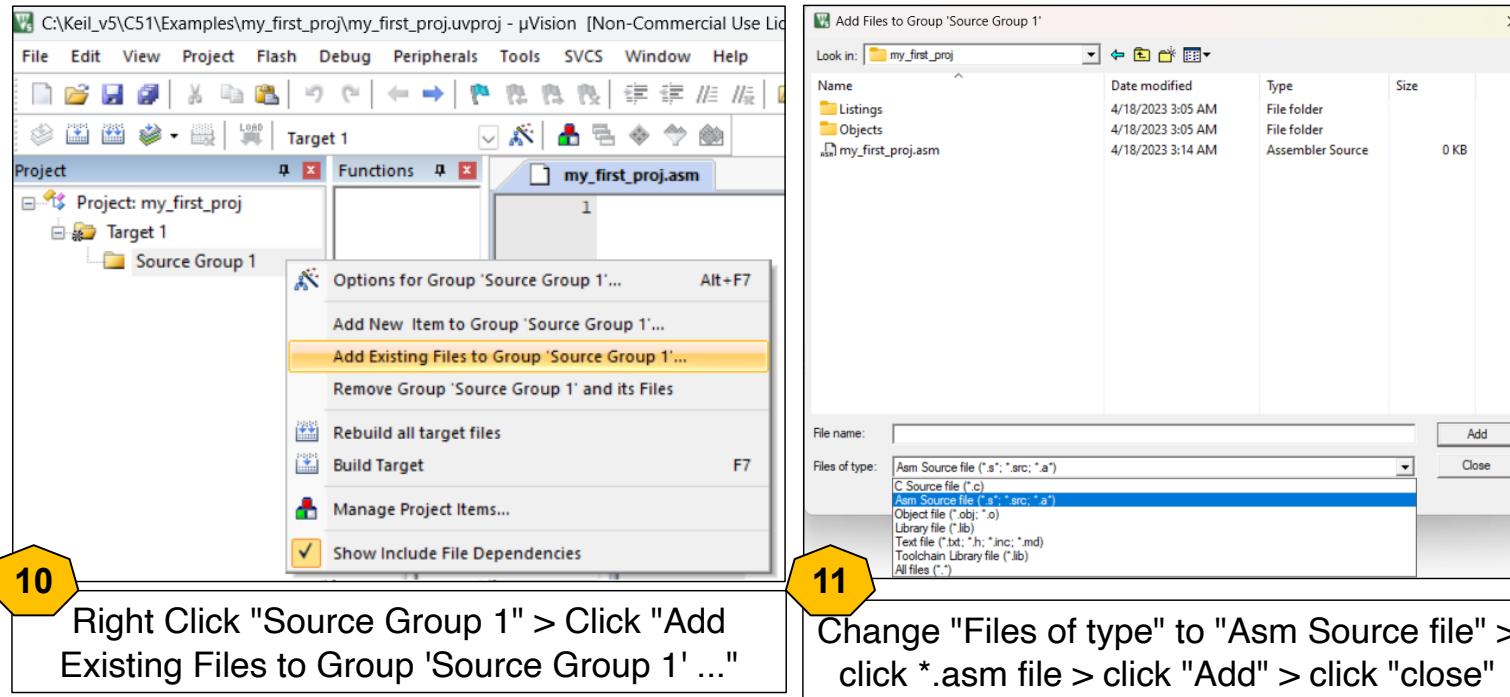
# KEIL TUTORIAL



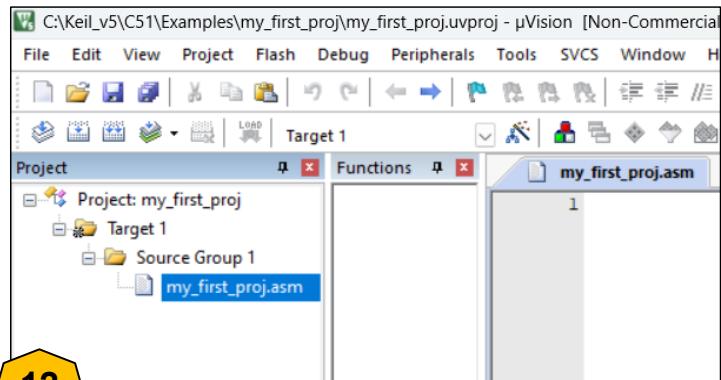
# KEIL TUTORIAL



# KEIL TUTORIAL

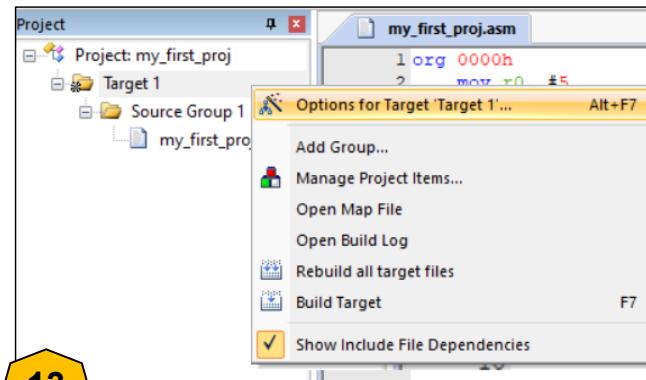


# KEIL TUTORIAL



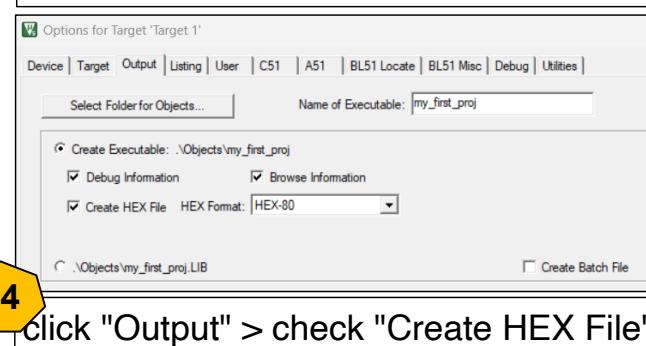
12

Our assembly file is added with the project. Let's write our first 8051 program which is to find factorial of a number :'). For 1st exercise we will write an assembly code and for 2nd exercise we will practice with a C program.



13

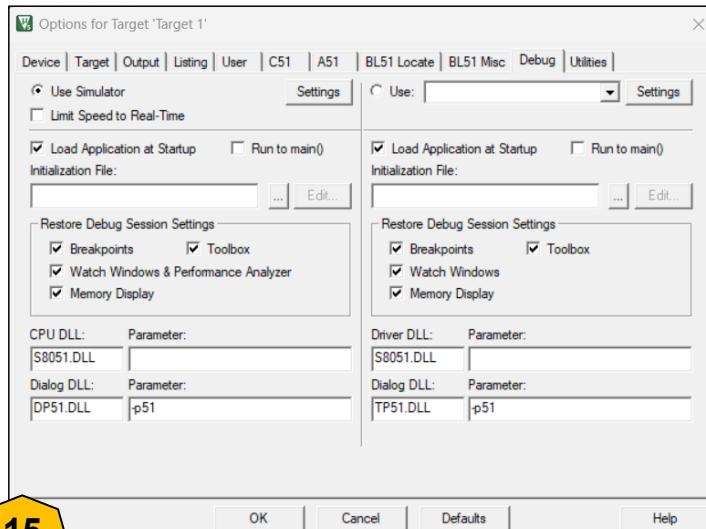
Right click "Target 1" > click "Options for Target 'Target 1'..."



14

click "Output" > check "Create HEX File"

# KEIL TUTORIAL



Un-check "Run to main()"(left side) as we are not running a C program > Click "OK"

```

my_first_proj.asm
1 org 0000h
2 mov r0, #5
3 mov A, r0
4 acall fact
5 fact: dec r0
6 cjne r0, #1, calculation
7 sjmp stop
8 calculation:
9 mov b, r0
10 mul ab
11 acall fact
12 stop:
13 end

```

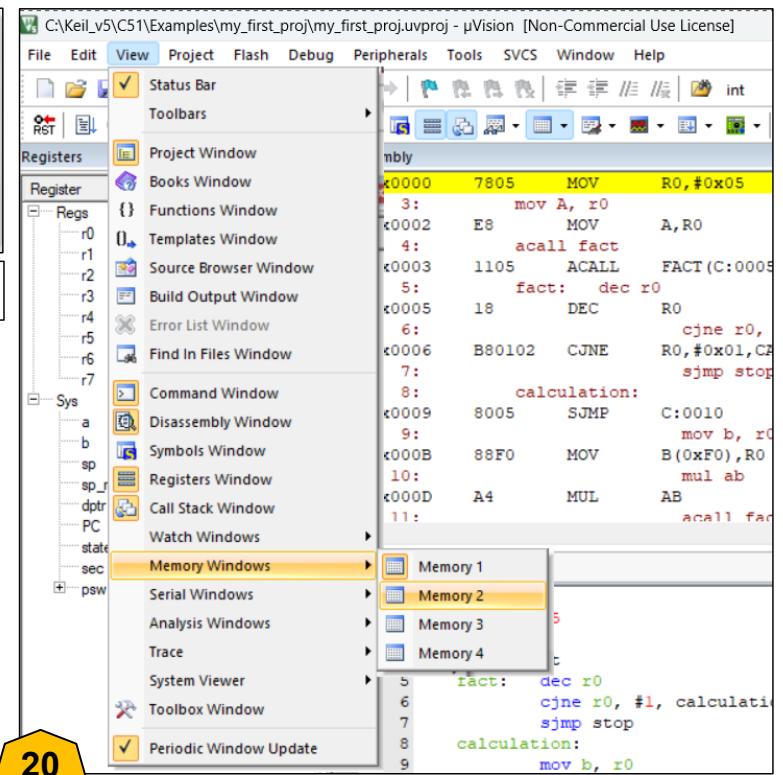
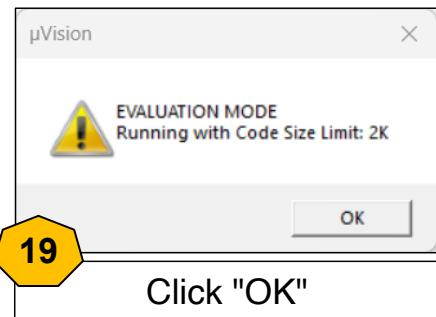
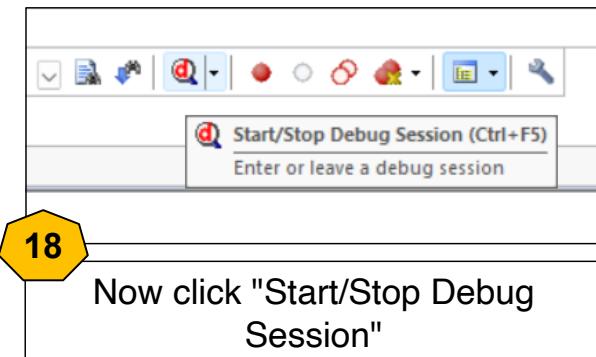
click "Save" > click "Translate" > click "Build"

Build Output

Build started: Project: my\_first\_proj  
Build target 'Target 1'  
linking...  
Program Size: data=8.0 xdata=0 code=16  
".\Objects\my\_first\_proj" - 0 Error(s), 0 Warning(s).  
Build Time Elapsed: 00:00:00

We should get 0 error and 0 warning after "Translate" and "Build"

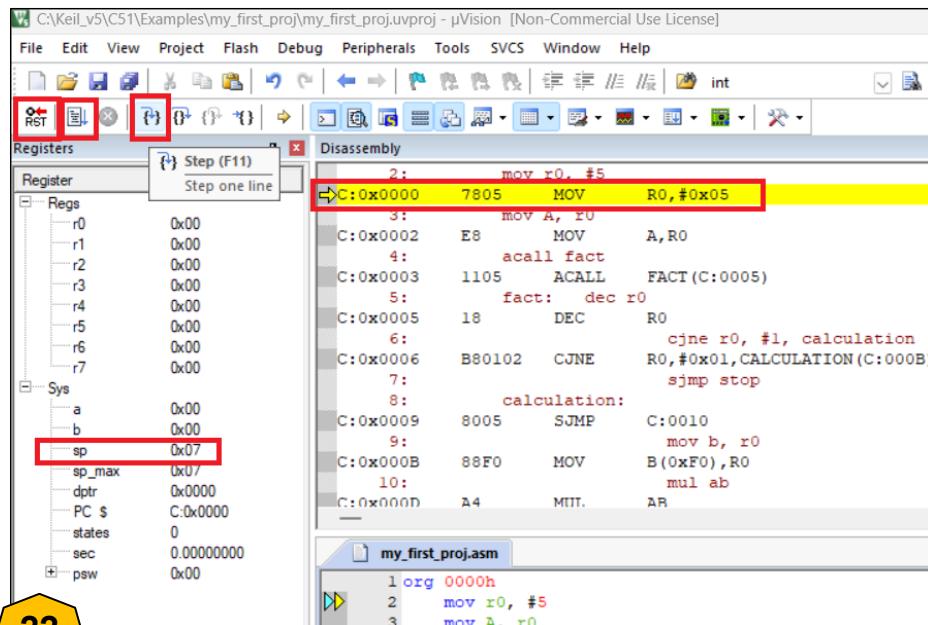
# KEIL TUTORIAL



21 In "Memory 1" "Address" type "D:0x00" and it will show all the internal data memory (256 byte). In "Memory 2" "Address" type "C:0x0000" and it will show the External memory (ROM) and the hex format machine codes.

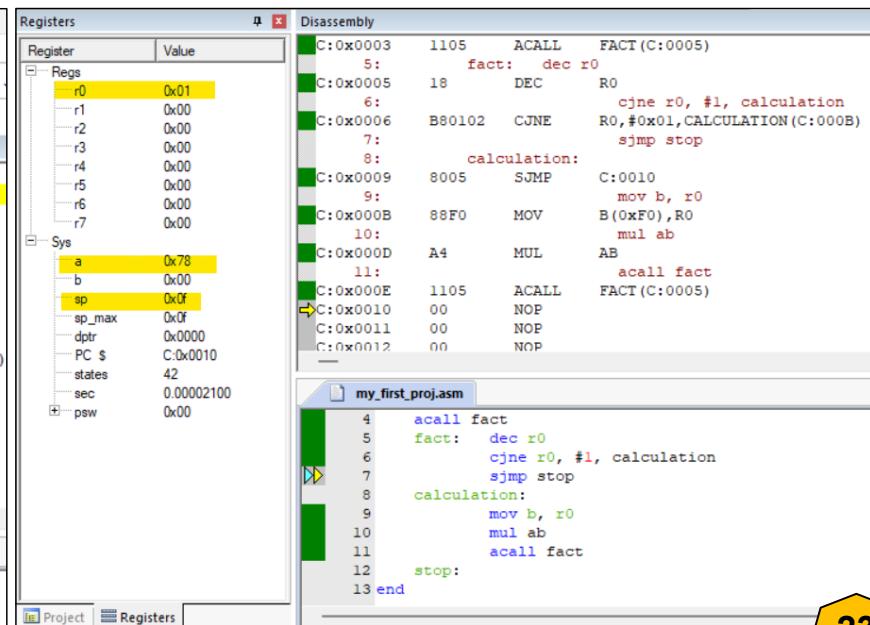
20 click "View" and you can Memory window to observe RAM and ROM data.

# KEIL TUTORIAL



22

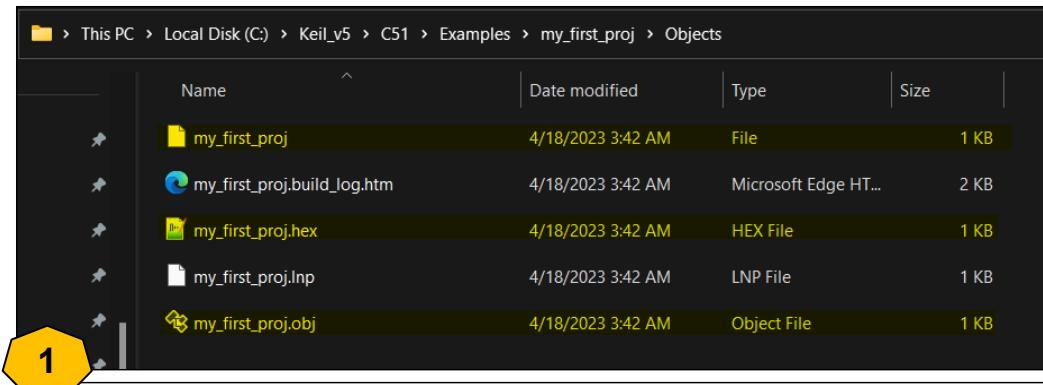
Continuously click on "Step" to observe the data flow among registers and reach the final output. Note that the current "sp" stack pointer is at "0x07" address, and it will store the return address after the function call on top of it.



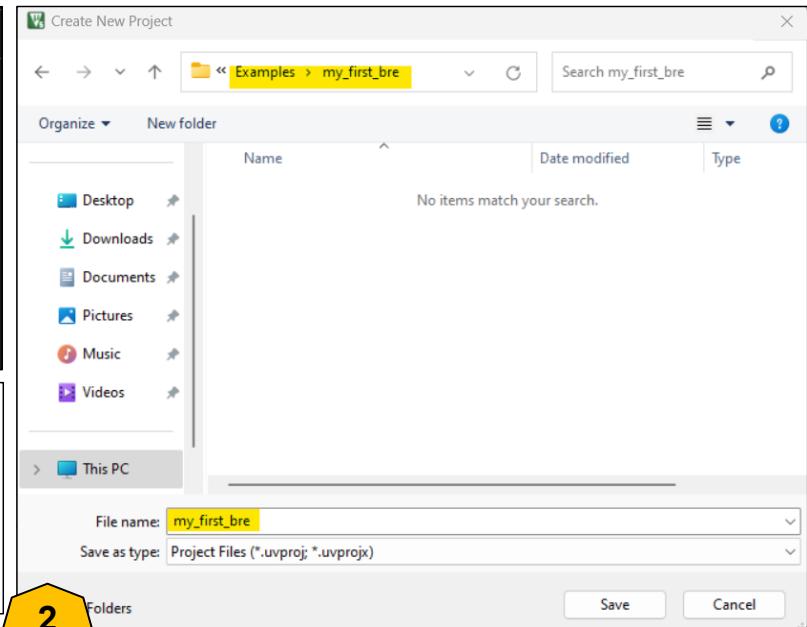
23

The final output is in accumulator register and the value is in hex "0x78" which is equivalent to 120 in decimal. Repeat step 18 to close the debugger window. Close the project from "Project" window.

# REVERSE ENGINEERING USING KEIL:



In the "Objects" folder inside the project directory we should see all the available output files. File "my\_first\_proj" is the compiled binary file and ready to load into the memory of AT89C51. Now we will observe how to load this binary file in Keil and extract information using disassembler.



Repeat step 1,2,3,4, and 5 to create a new project for reverse engineering.

# REVERSE ENGINEERING USING KEIL:

The screenshot shows the Keil MDK-ARM interface with several windows open:

- Registers**: Shows the CPU registers (r0-r7, Sys) with their current values.
- Disassembly**: Shows assembly code starting at address C:0x0000, all of which is NOP (No Operation).
- Memory 2**: Shows memory starting at address C:0x0000, filled with zeros.
- Command**: Shows the command line interface with the following history:

```
Running with Code Size Limit: 2K
Load "C:\\Keil_v5\\C51\\\\Examples\\\\my_first_bre\\\\Objects\\\\my_first_bre"
^
*** error 56: can't open file
```

A new command is entered:  
load C:\\Keil\_v5\\C51\\Examples\\my\_first\_proj\\Objects\\my\_first\_proj  
<filespec>
- Project**: Shows the project structure.

**Step 3** is indicated by a yellow hexagon on the left side of the interface.

**Step 4** is indicated by a yellow hexagon on the right side of the Command window.

**Text Box (Step 3):** Repeat step 18 and 19 to enter the debugging mode. Ignore the error.

**Text Box (Step 4):** Use "load" command to read the binary file from previous "\*project\*/Objects" directory. If the tool load the file successfully, it will give no error.

# REVERSE ENGINEERING USING KEIL:

Memory 2

Address	Value
C:0x0000:	78 05 E8 11 05 18 B8 01 02 80 05 88 F0 A4 11 05 0
C:0x0025:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x004A:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x006F:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x0094:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x00B9:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x00DE:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x0103:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x0128:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
C:0x014D:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
x0172:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0

5

In the external memory we should observe the hex machine code. Now we will disassemble it.

Command

```
Running with Code Size Limit: 2K
Load "C:\\Keil_v5\\C51\\Examples\\my_first_bre\\Objects\\my_first
^
*** error 56: can't open file
load C:\\Keil_v5\\C51\\Examples\\my_first_proj\\Objects\\my_first_proj

>U C:0x0000
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
```

6

Using "U <starting memory location>" we can disassemble the binary file.

# REVERSE ENGINEERING USING KEIL:

The screenshot shows the Keil MDK-ARM interface with three main windows:

- Registers** window: Shows the state of various registers. The **r0** register is highlighted with a blue selection bar, containing the value **0x01**.
- Disassembly** window: Displays assembly code. Key instructions include:
  - MOV R0, #0x05
  - ACALL FACT (C:0005)
  - DEC R0
  - CJNE R0, #0x01, CALCULATION(C:000B)
  - SJMP C:0010
  - MOV B(0xF0), R0
  - MUL AB
  - ACALL FACT (C:0005)
  - NOP (multiple instances)
- Command** window: Shows the command-line interface output. It includes error messages and a command to save the program state:

```
Running with Code Size Limit: 2K
Load "C:\\Keil_v5\\C51\\Examples\\my_first_bre\\Objects\\my_first
^
*** error 56: can't open file
load C:\\Keil_v5\\C51\\Examples\\my_first_proj\\Objects\\my_first_proj
U C:0x0000
*** error 65: access violation at C:0x0010 : no 'execute/read' pe
> SAVE op.hex 0x0000, 0x0010
```

We should observe the above assembly code in "Disassembly" window. After running the assembly file, in the accumulator register we got "0x78"

7

8

Let's save the program state in hex format from external code region.

**Congratulation! You successfully completed the reverse engineering!!**

O

• W