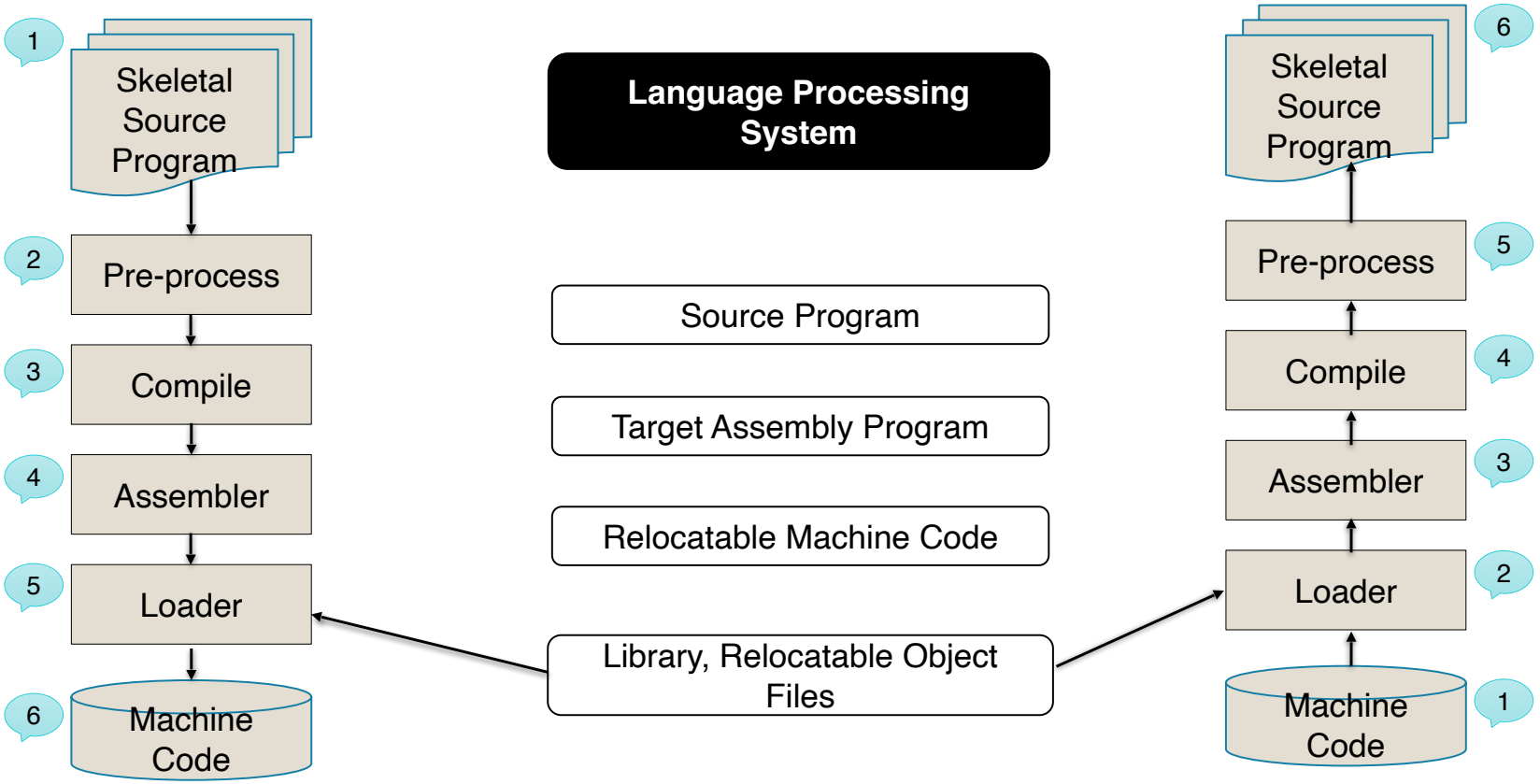
EECS 388:
Embedded Systems

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.

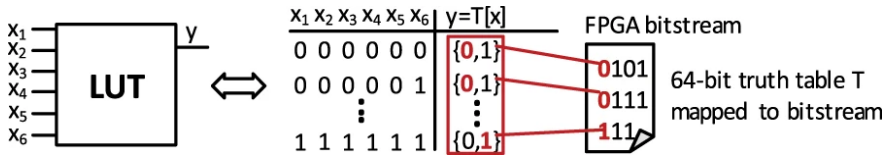
- Understand the properties of machine code/object code.
- Each architecture has fixed set of opcode, operand and register naming.
- Can be Static or Dynamic.

- 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

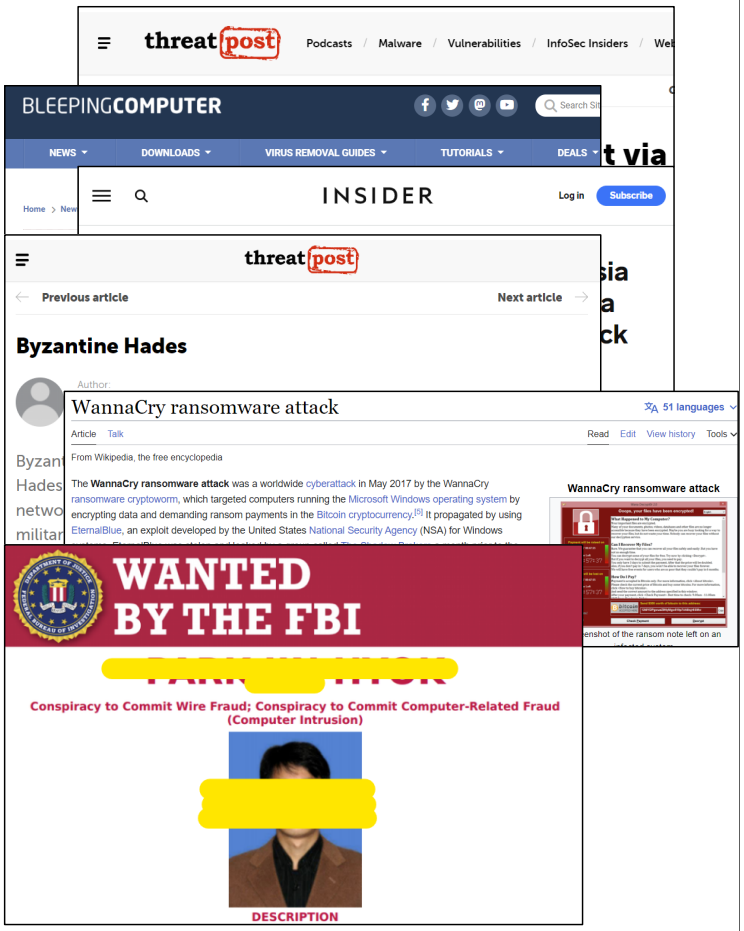
MicroController 4 All: Session 5 : 8051 Addressing Modes



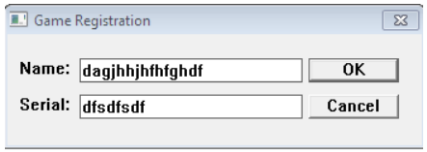
Bitstream Fault Injections (BiFI)–Automated Fault Attacks Against SRAM-Based FPGAs (computer.org)

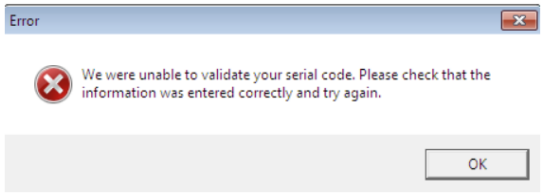
WHY REV. ENG. ?

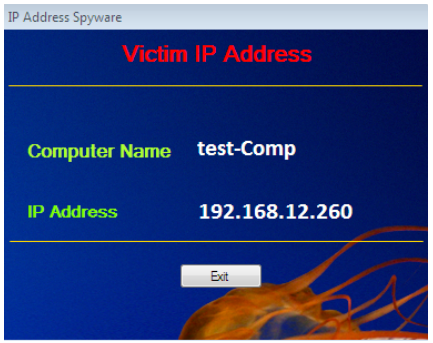
- Legitimate purposes
  - Understand the program
  - Identify vulnerabilities/weakness
  - Analyze security vulnerabilities
- Malicious purposes
  - Unauthorized overproduction / IP piracy
  - Develop malware / create malicious version
  - Trojan insertion



EXAMPLE

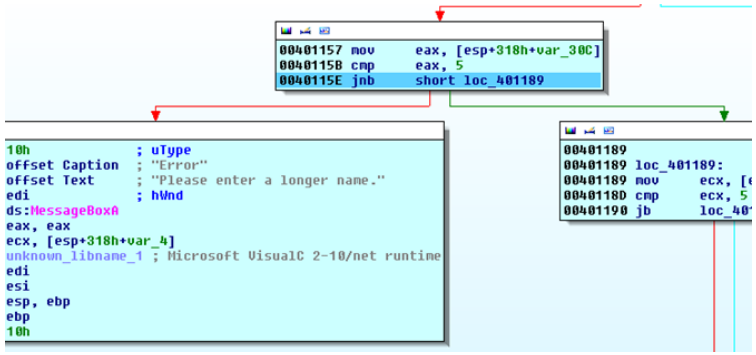






Step 1

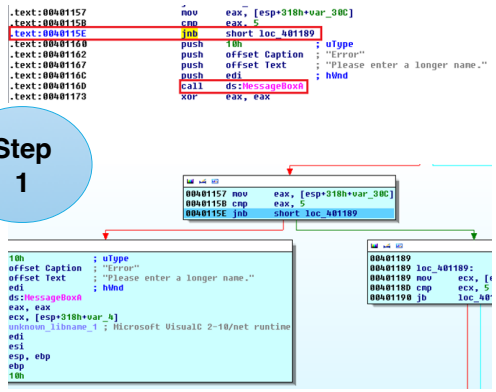
```
.text:00401157                mov     eax, [esp+318h+var_30C]
.text:0040115B                cmp     eax, 5
.text:0040115E                jnb     short loc_401189
.text:00401160                push    10h             ; uType
.text:00401162                push    offset Caption  ; "Error"
.text:00401167                push    offset Text     ; "Please enter a longer name."
.text:0040116C                push    edi             ; hWnd
.text:0040116D                call    ds:MessageBoxA
.text:00401173                xor     eax, eax
```



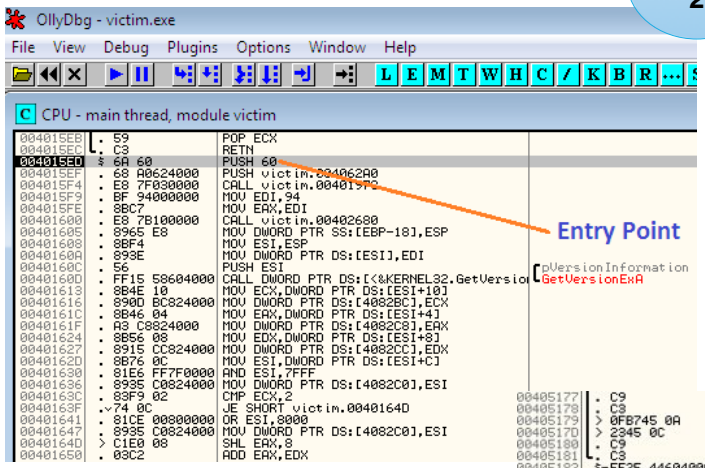
Tool used: IDA Pro

EXAMPLE

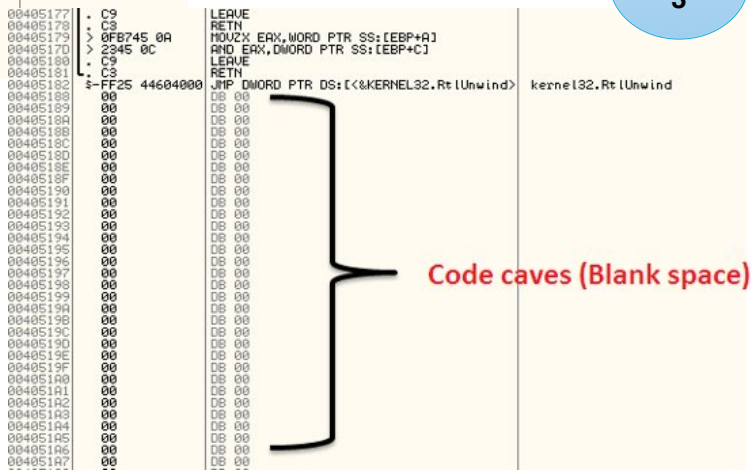
Step 1



Step 2



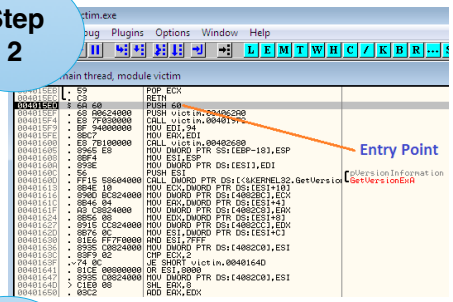
Step 3



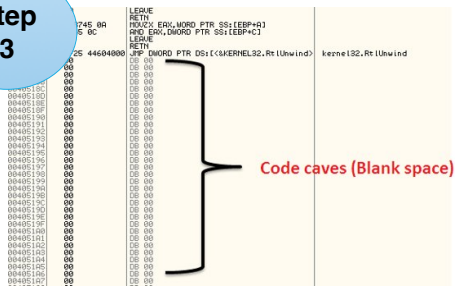
Tool used: OllyDbg

EXAMPLE

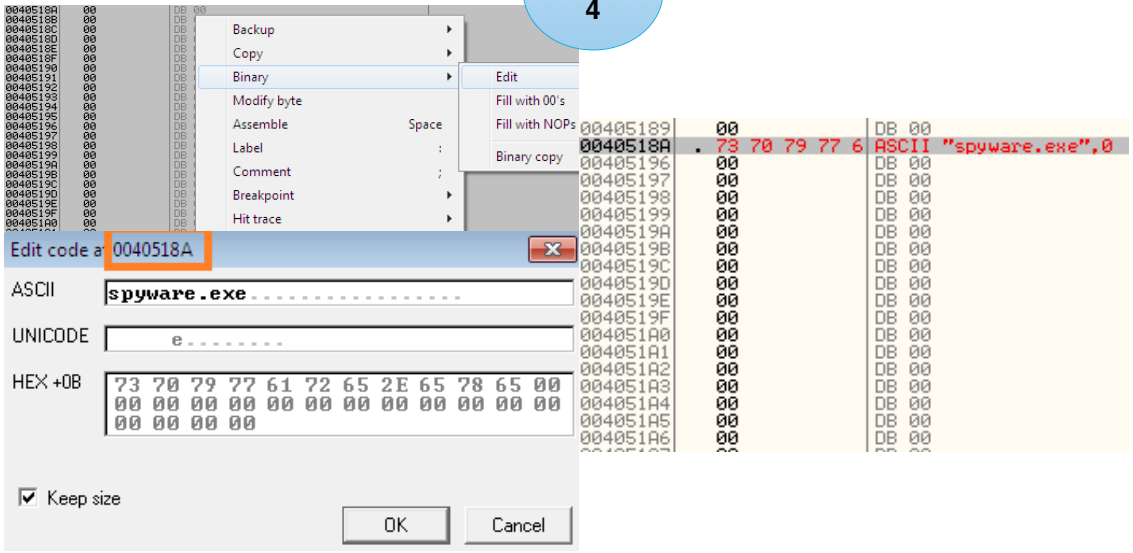
Step 2



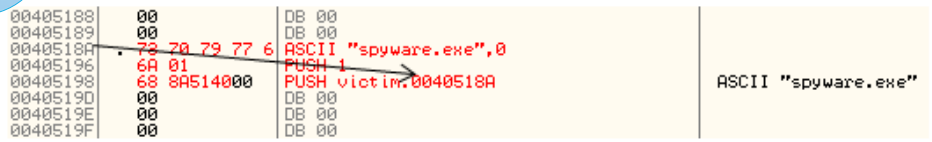
Step 3



Step 4

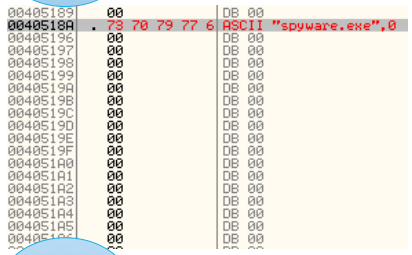


Step 5

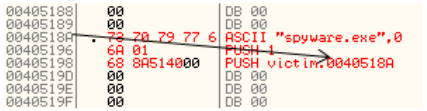


EXAMPLE

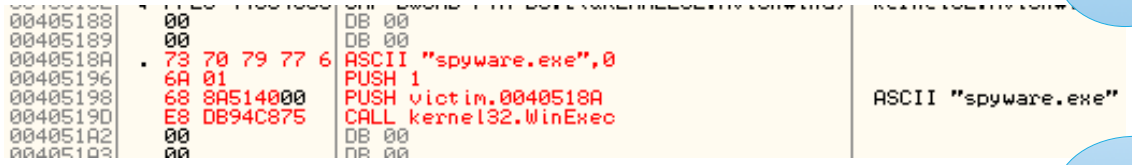
Step 4



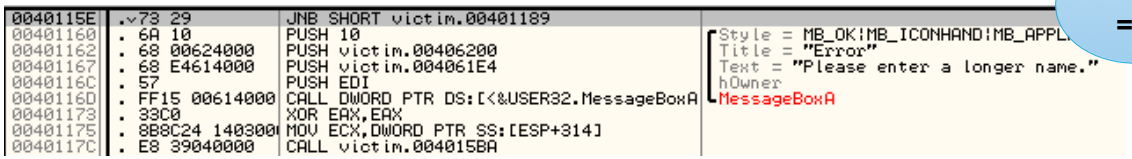
Step 5



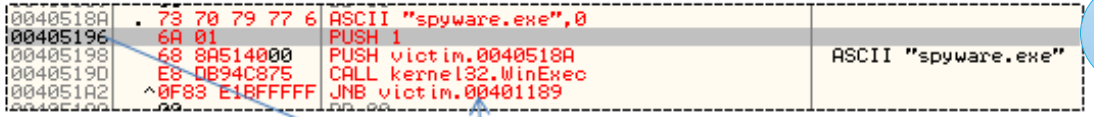
Step 6



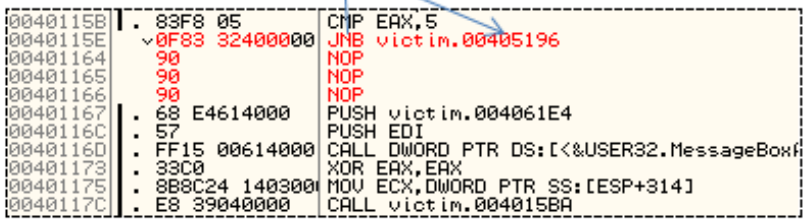
Step 7 = 1



Step 8



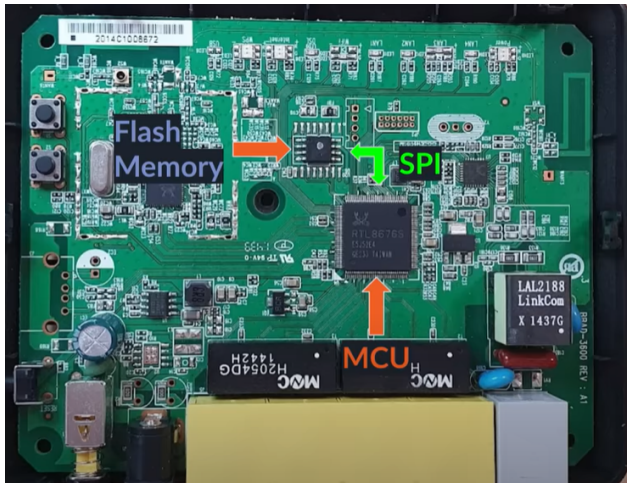
(This was JNB SHORT 00401189 earlier before alter)



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

REV. ENG. IN EMBEDDED DEVICES

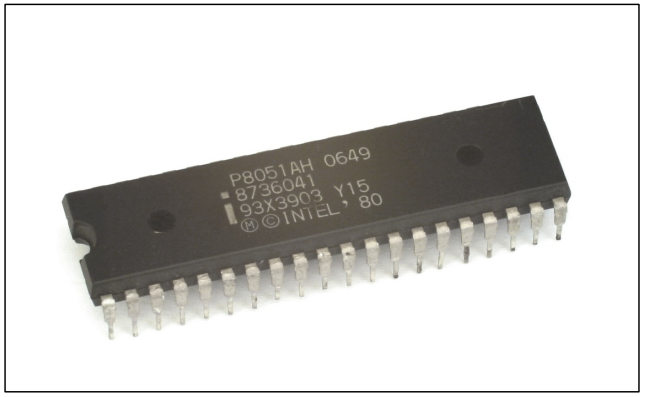
- Loading a program onto an embedded device required following steps:
  - Writing the program in behavior level i.e. C or C++
  - Compile the code into a machine readable format
  - Load the binary/executable image in device memory
- Can an attacker with physical access to the device steal the binary 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.

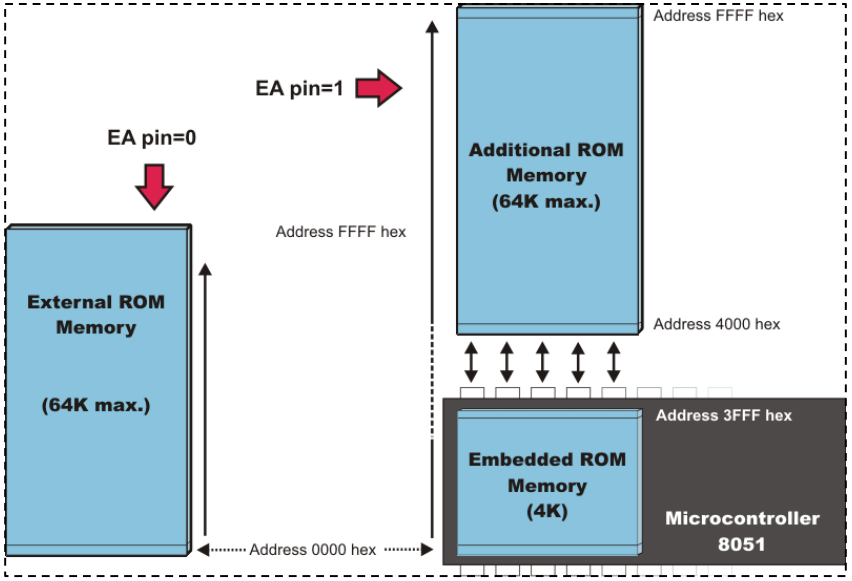
INTRO TO 8051 MCU:

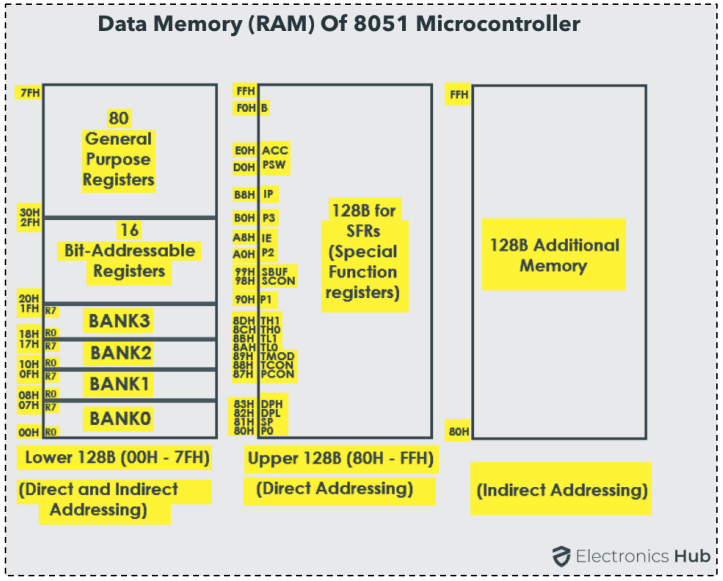
- 8-bit architecture introduced by intel in 1980s.
- 8-bit CPU, 4 KB of ROM, 128 bytes of RAM and 32 I/O pins
- Simple instruction set and can execute one instruction per cycle
- Can be programmed in assembly language or high-level languages such as C
- Still used today for embedded applications that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



Source: Intel 8051 - Wikipedia

MEMORY ORGANIZATION





Source: 8051 Memory Organization (mikroe.com)

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.

INTRO TO 8051 MCU ISA:

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

INTRO TO 8051 MCU ISA:

OPCODE	USAGES	Details
CJNE	CJNE OP1, OP2, LOC OP1 != OP2, LOC⇔ JUMP LOC OP1 == OP2 ⇔ 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 ⇔ JUMP LOC and OP1-- OP1 == 0 ⇔ 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}
Immediate Addressing Mode ⇒ MOV A, #5h # is followed by hex data	Here, Source = Data Note: ⇒ MOV A, #Fh ⇔ ERROR! , ⇒ MOV A, #0Fh ⇔ CORRECT!
Register Addressing Mode ⇒ MOV A, R1 ⇒ MOV R2, R1 ⇔ ERROR!	Here, Source = Register 8051 is an ACCUMULATOR-based architecture.
Direct or Absolute Addressing 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.
Indirect Addressing ⇒ 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
Index Addressing ⇒ 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
Program Counter	Keeps track of where the processor is in its execution of a program.
Accumulator	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.
Stack Pointer	Holds the memory address of the top of the stack
Instruction Register	Holds the current instruction that the processor is executing
Address Register	Holds the address of the next instruction to be executed
Index Register	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:

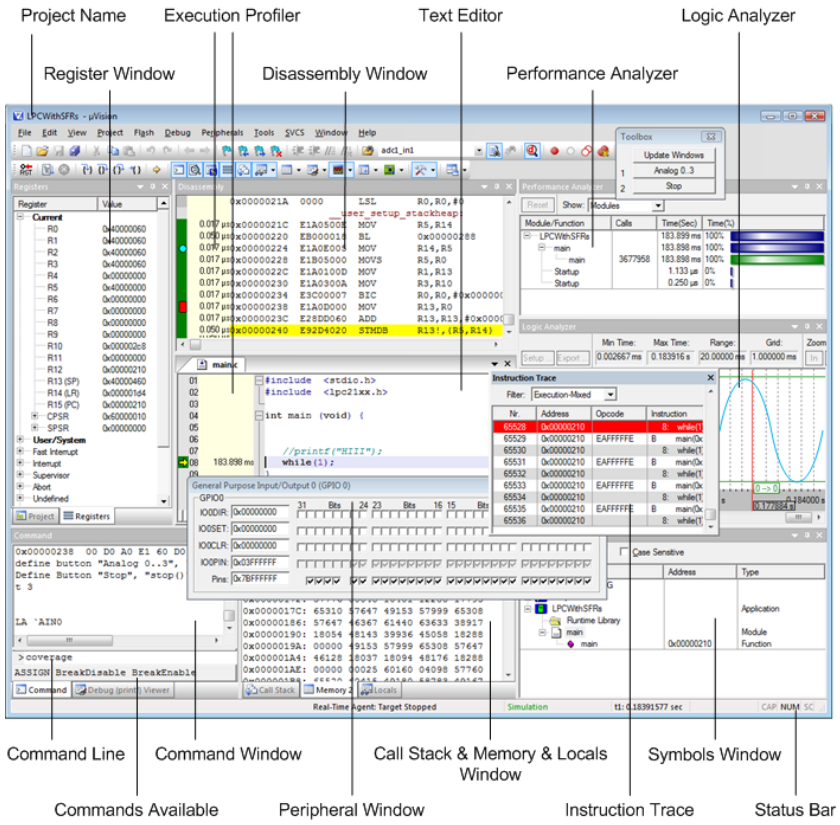
- KEIL C51 is a cross-compiler for the 8051 microcontroller family.
- It supports a wide range of 8051 variants and extensions, such as code banking, memory models, and bit-addressable data types.
- IDE with a project manager, editor, debugger, and simulator.
- It includes a library of standard and vendor-specific 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.

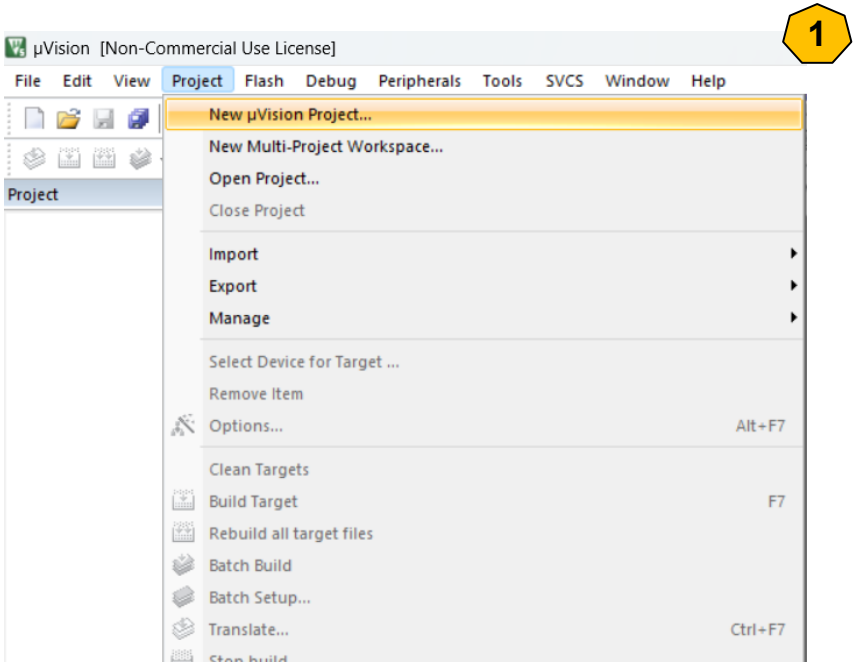
Download Link: Keil Product Downloads

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

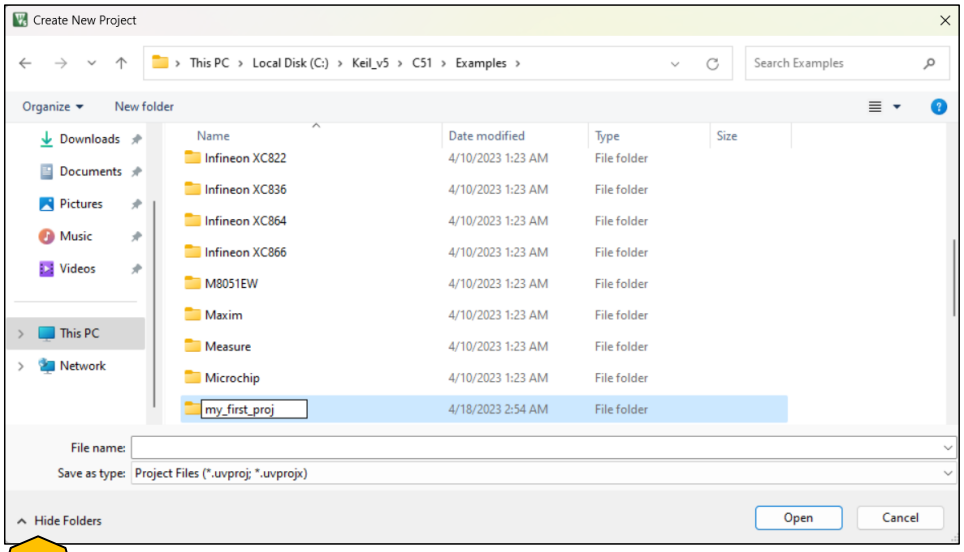
KEIL TUTORIAL





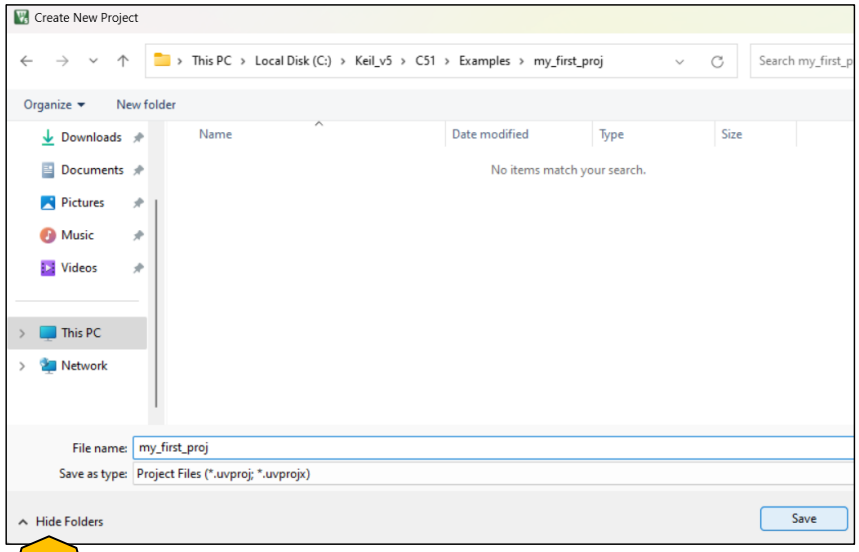
Project > New uVision Project

KEIL TUTORIAL



2

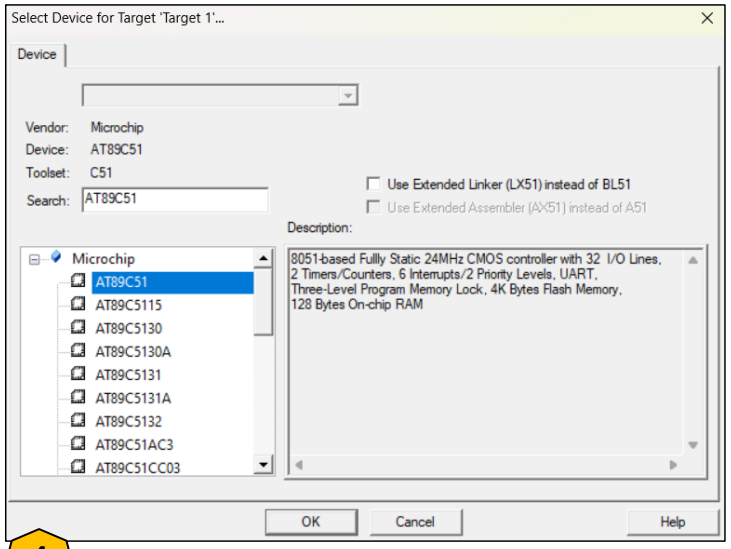
Click "New folder" > give a name > click "Open"



3

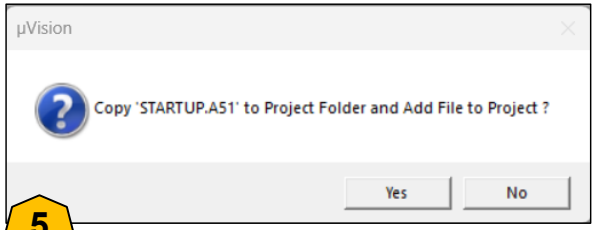
Same name as folder > click "Save"

KEIL TUTORIAL



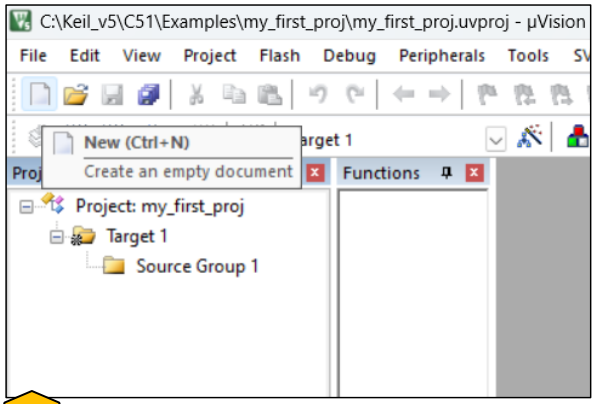
4

Search "AT89C51" > Select from "Microchip" > click "OK"



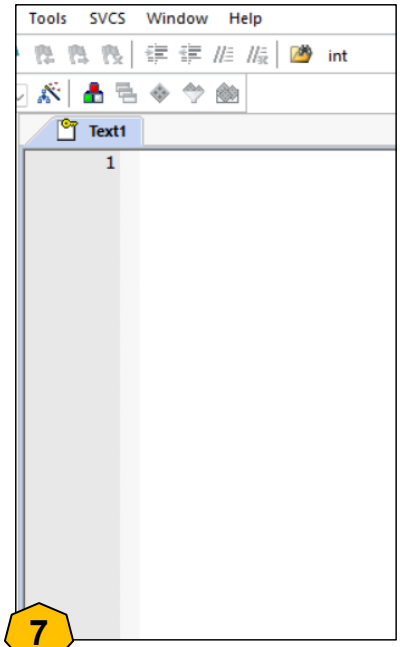
5

click "No"



6

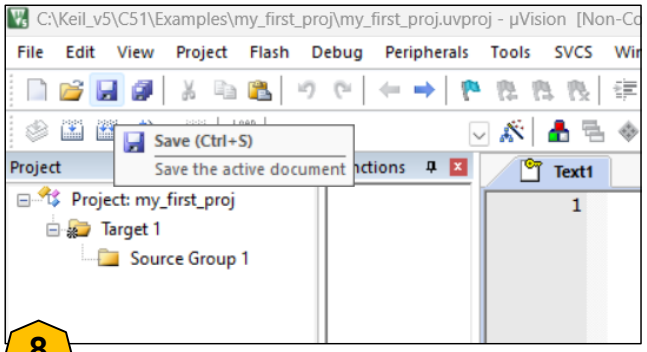
click "New" or "Ctrl+N"



7

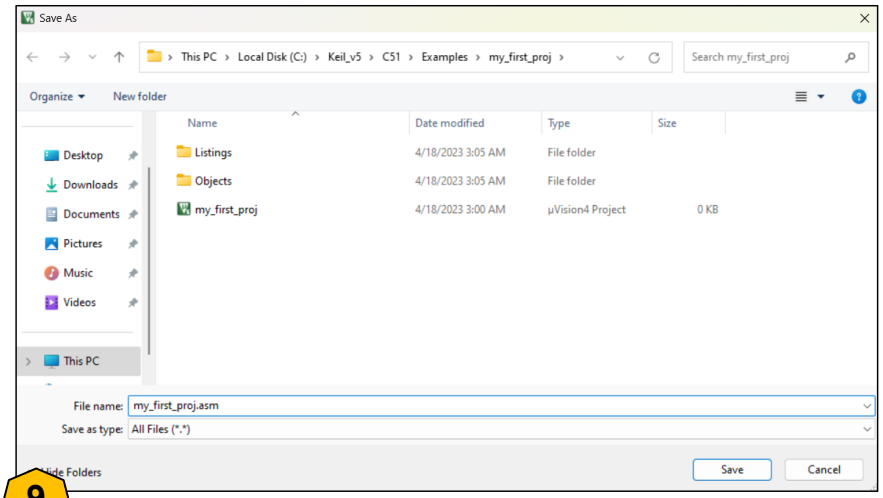
A new text window will appear...

KEIL TUTORIAL



8

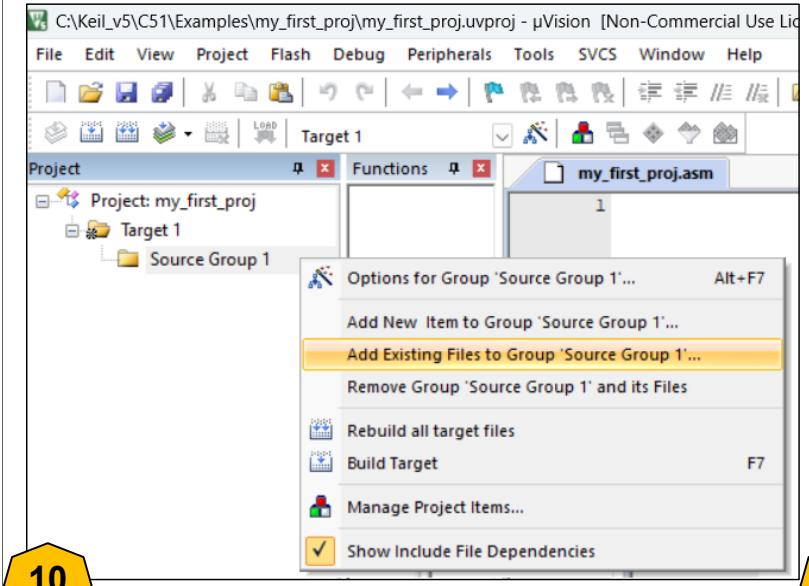
click "Save"



9

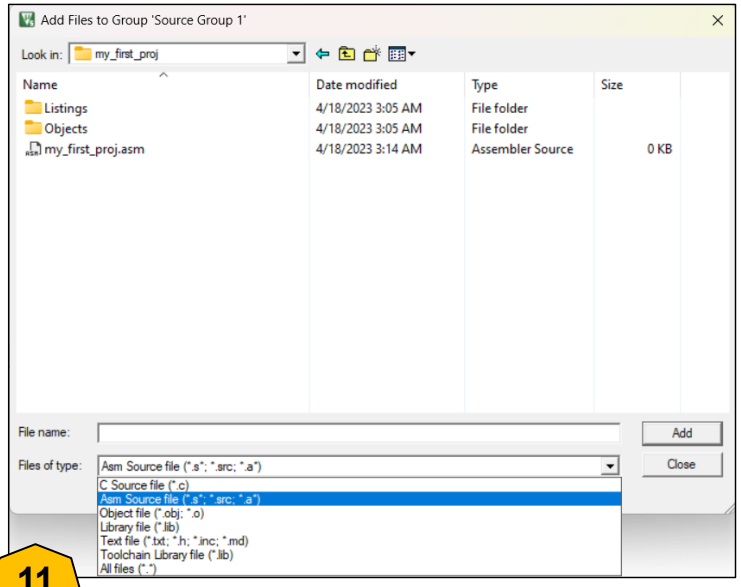
Same name as folder with extension .asm > click "Save"

KEIL TUTORIAL



10

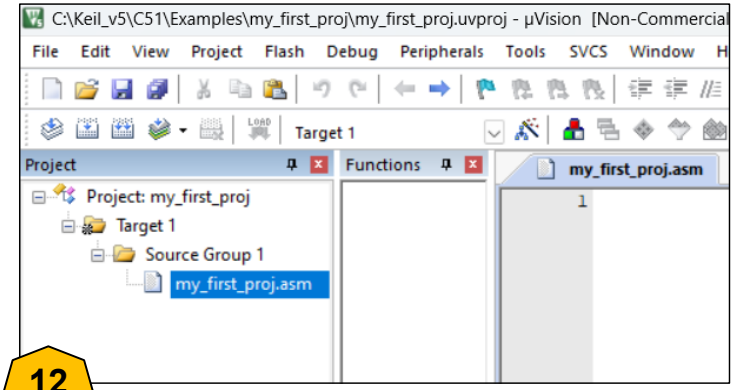
Right Click "Source Group 1" > Click "Add Existing Files to Group 'Source Group 1' ..."



11

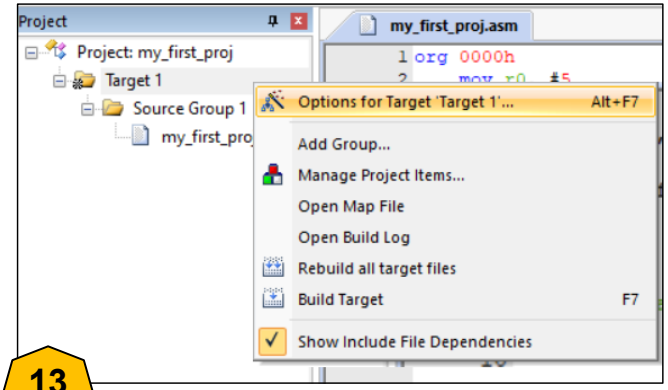
Change "Files of type" to "Asm Source file" > click *.asm file > click "Add" > click "close"

KEIL TUTORIAL



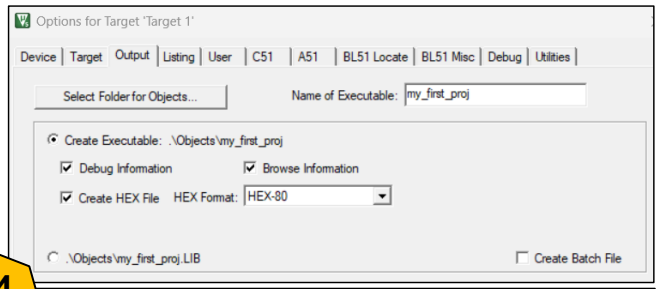
12

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

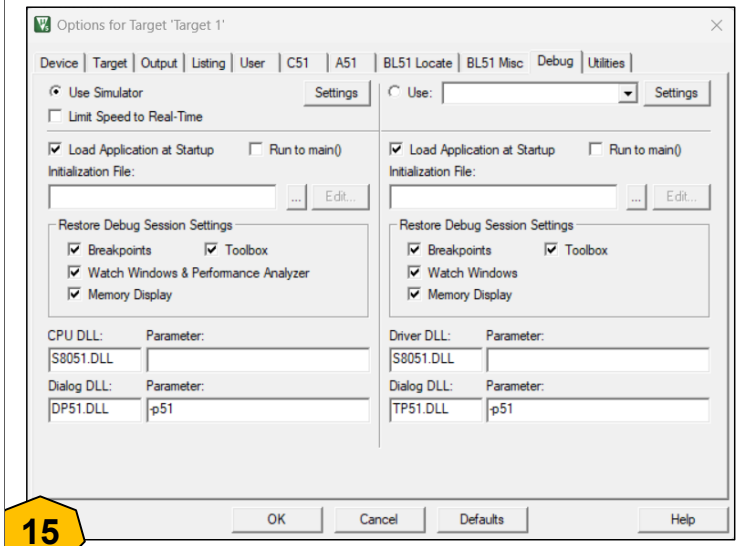
Right click "Target 1" > click "Options for Target 'Target 1'..."



14

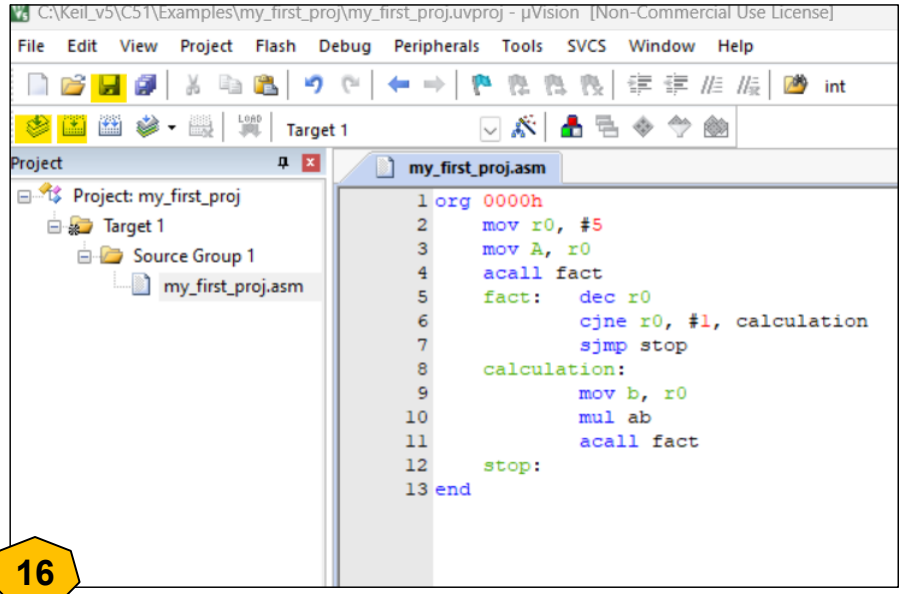
click "Output" > check "Create HEX File"

KEIL TUTORIAL



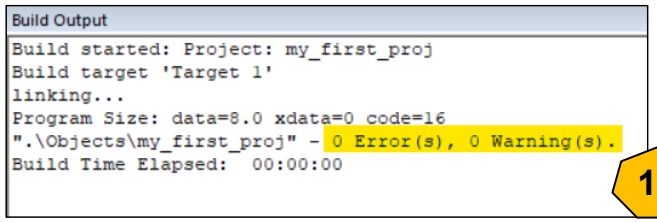
15

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



16

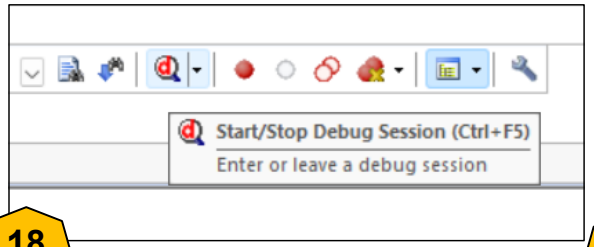
click "Save" > click "Translate" > click "Build"



17

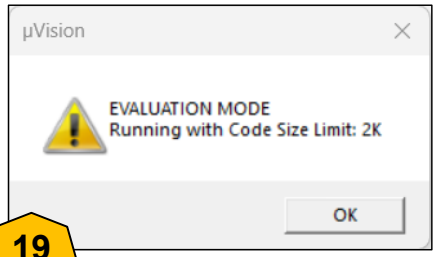
We should get 0 error and 0 warning after "Translate" and "Build"

KEIL TUTORIAL



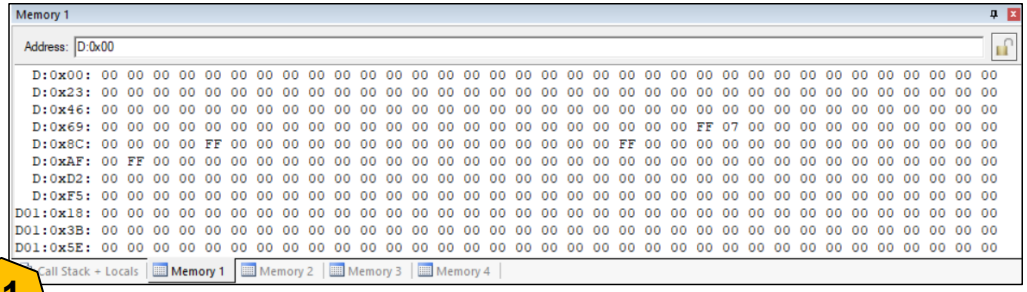
18

Now click "Start/Stop Debug Session"



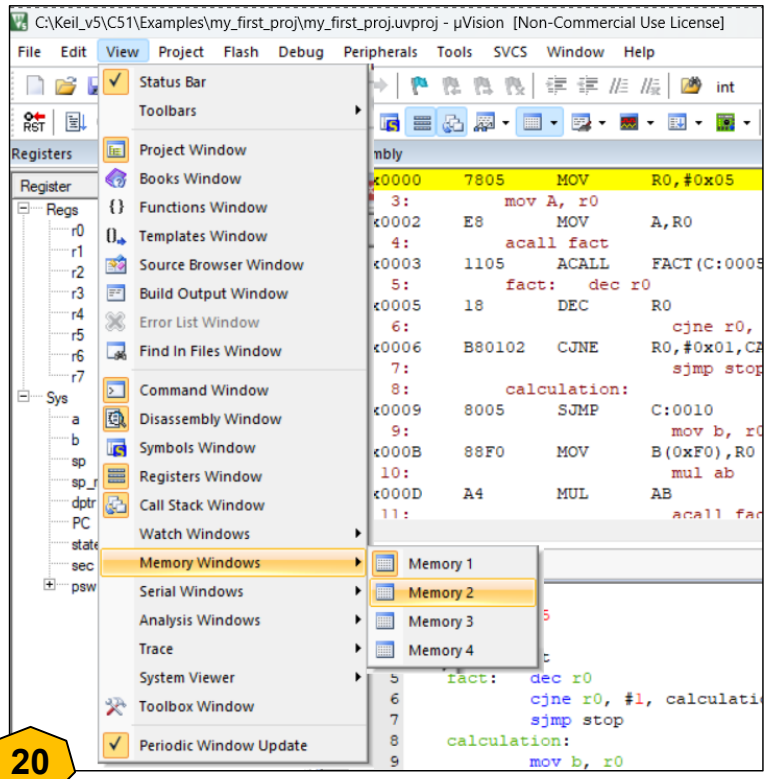
19

Click "OK"



21

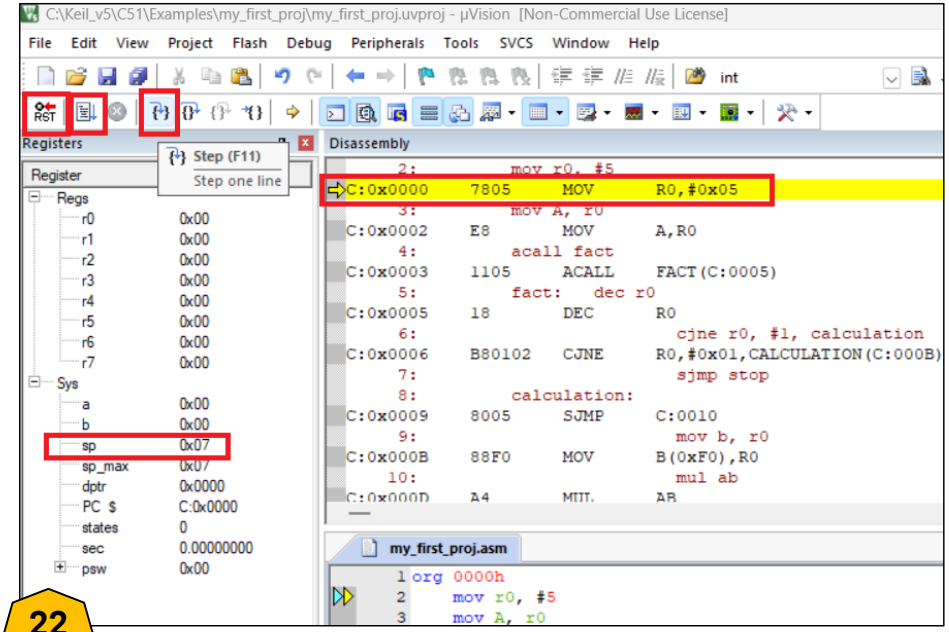
In "Memory 1" "Address" type "D:0x00" and it will show all the internal data memory (256 byte). In "Memory 2" "Address" type "C:0x0000" and it will show the External memory (ROM) and the hex format machine codes.



20

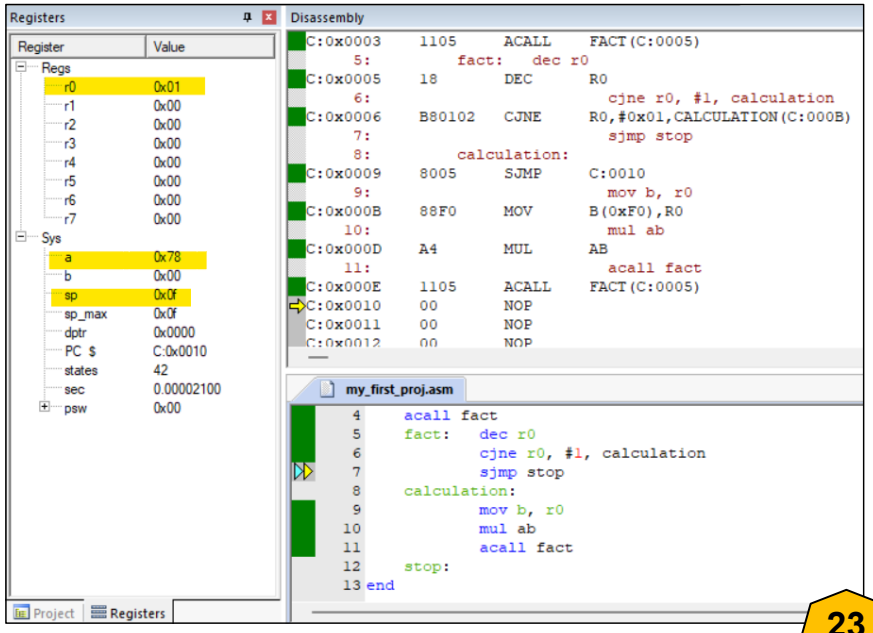
click "View" and you can Memory window to observe RAM and ROM data.

KEIL TUTORIAL



22

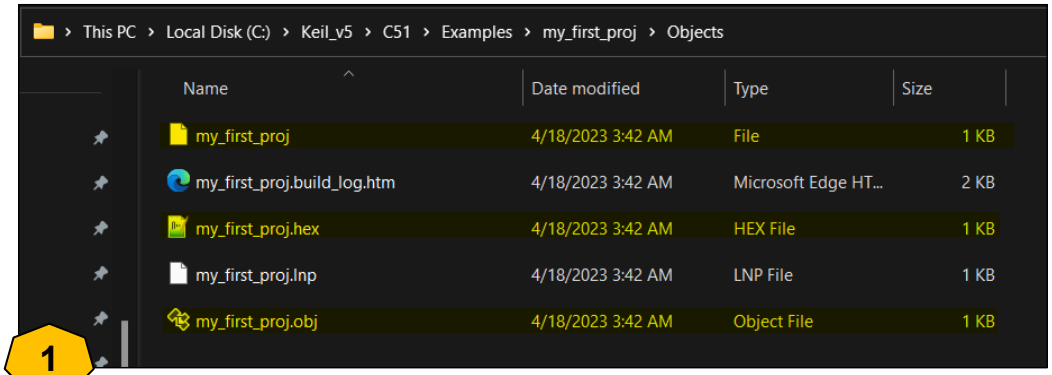
Continuously click on "Step" to observe the data flow among registers and reach the final output. Note that the current "sp" stack pointer is at "0x07" address, and it will store the return address after the function call on top of it.



23

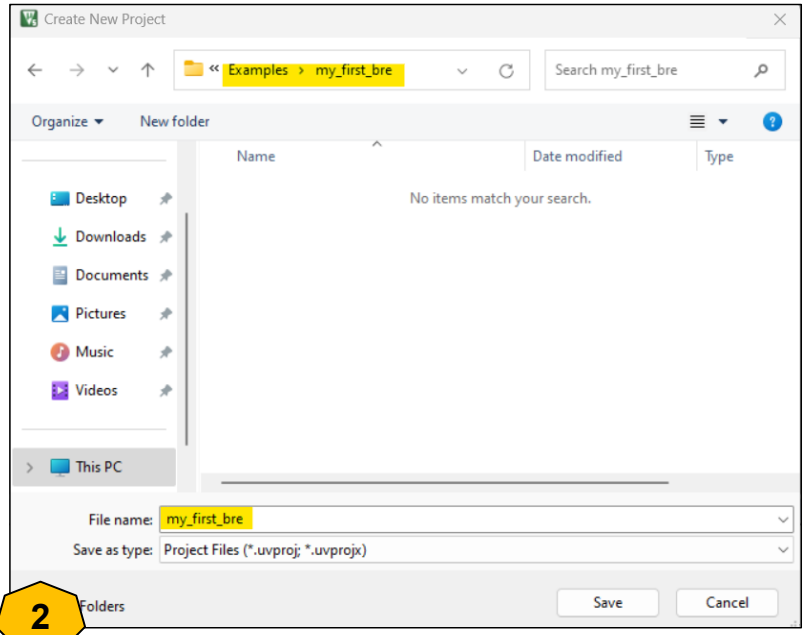
The final output is in accumulator register and the value is in hex "0x78" which is equivalent to 120 in decimal. Repeat step 18 to close the debugger window. Close the project from "Project" window.

REVERSE ENGINEERING USING KEIL:



1

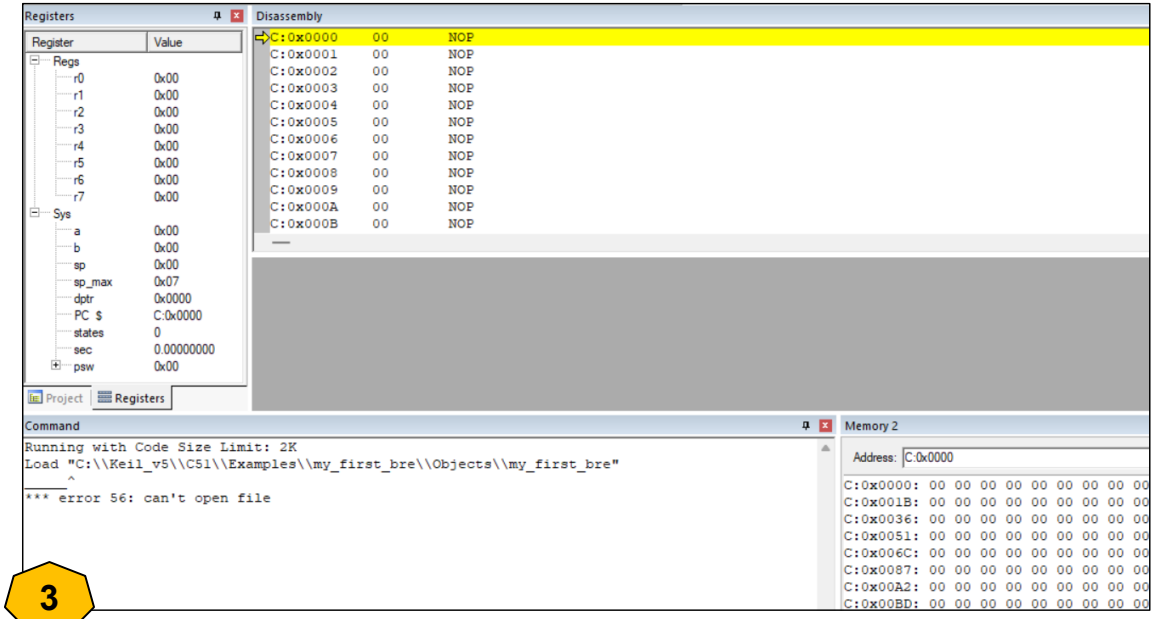
In the "Objects" folder inside the project directory we should see all the available output files. File "my_first_proj" is the compiled binary file and ready to load into the memory of AT89C51. Now we will observe how to load this binary file in Keil and extract information using disassembler.



2

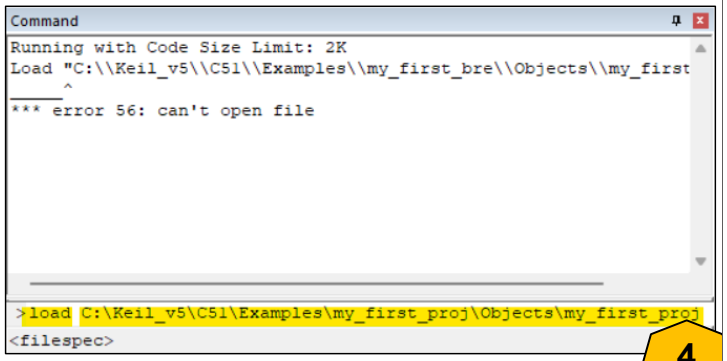
Repeat step 1,2,3,4, and 5 to create a new project for reverse engineering.

REVERSE ENGINEERING USING KEIL:



3

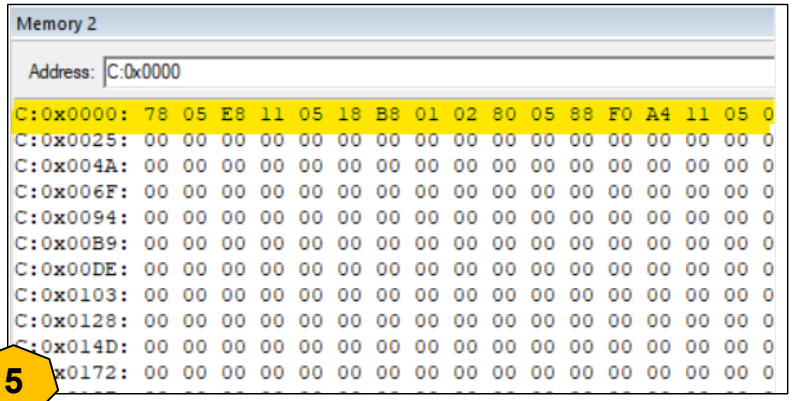
Repeat step 18 and 19 to enter the debugging mode. Ignore the error.



4

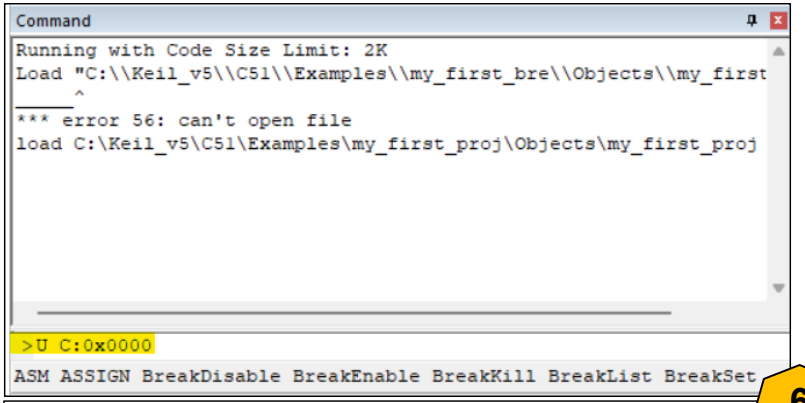
Use "load" command to read the binary file from previous "*project*/Objects" directory. If the tool load the file successfully, it will give no error.

REVERSE ENGINEERING USING KEIL:



5

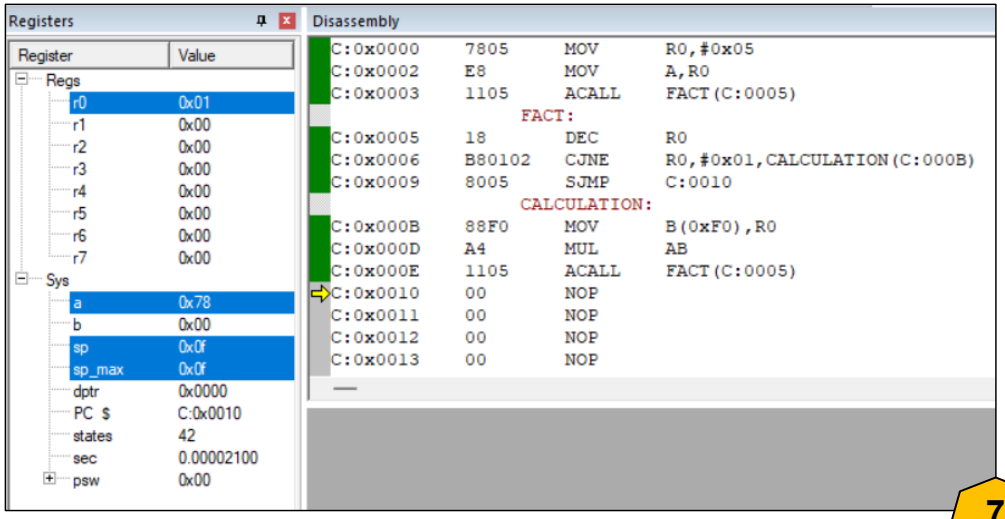
In the external memory we should observe the hex machine code. Now we will disassemble it.



6

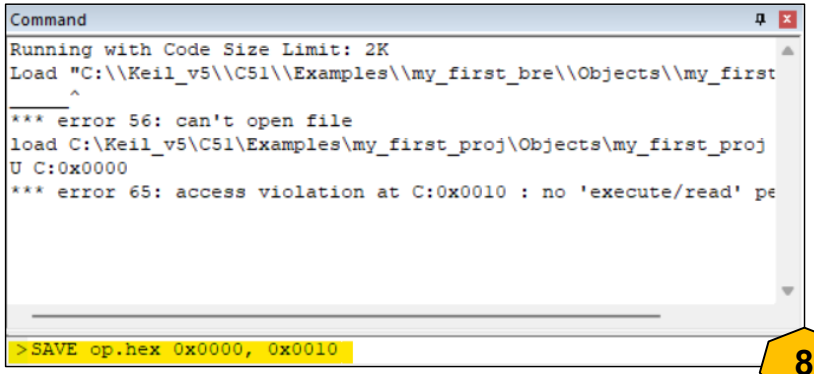
Using "U <starting memory location>" we can disassemble the binary file.

REVERSE ENGINEERING USING KEIL:



7

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



8

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

Congratulation! You successfully completed the reverse engineering!!

o

- W