

# **Keil uVision Debugging Programs**

**WEL Labs, IITB  
2016**

# Development Tools (Revision)





---

- **Coding – Editor => Entry of code into file(s)**
- **Translation – Assembler or Compiler**  
=> Generate machine code from source code
- **Execution check – using Debugger to verify operation of program ( on Simulator )**
- **Program – Programmer**  
=> Put machine code in the chip

**Single Point Solution – IDE e.g. Keil**

# Keil uVision IDE ...

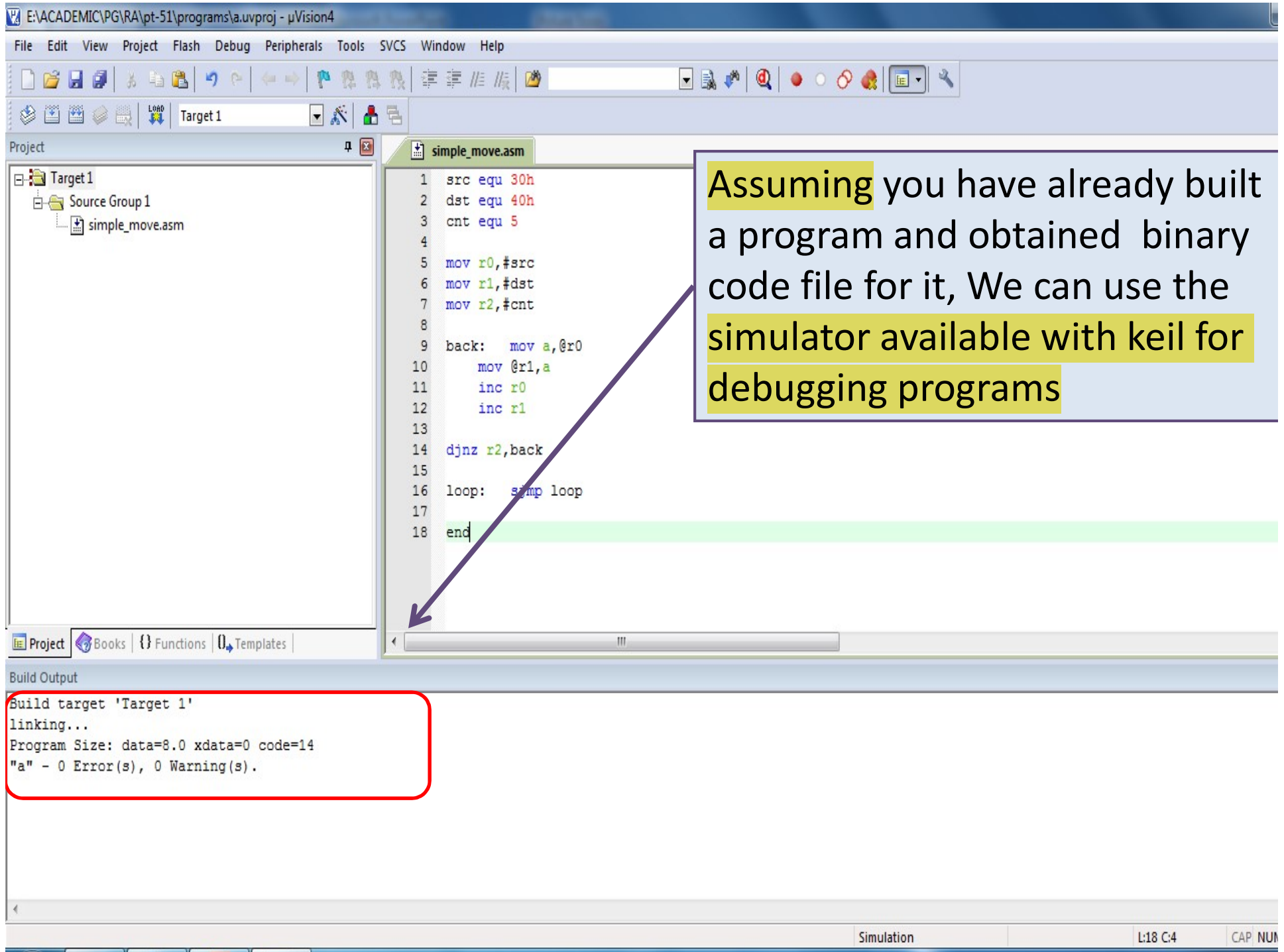
---

-  **Project** : A **collection of files related to a particular programming task.**
-  **Build** : The process in which **only the files modified since last build** are assembled/compiled for the chosen microcontroller device.
-  **Rebuild** : The process in which **all files are assembled/compiled** irrespective of their modification state.
-  **Debug** : The process of finding errors happening during program execution and removing them.

# Simulator and Debugger

---

- Simulation of microcontroller behavior while executing the user program
- The Debug mode UI allows the user to perform the following
  1. **Observe** microprocessor Registers, Memory, Ports and Peripherals
  2. **Place breakpoints** to stop simulation as specific instruction or on condition
  3. **Monitor code** under execution
  4. **Modify data** variables
  5. **Monitor timing** of execution



E:\ACADEMIC\PG\RA\pt-51\programs\simple\_move.asm - µVision4

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Start/Stop Debug Session Ctrl+F5

Reset CPU

Run F5

Stop F11

Step F10

Step Over Ctrl+F11

Step Out Ctrl+F10

Run to Cursor Line

Show Next Statement

Breakpoints... Ctrl+B

Insert/Remove Breakpoint F9

Enable/Disable Breakpoint Ctrl+F9

Disable All Breakpoints

Kill All Breakpoints Ctrl+Shift+F9

OS Support

Execution Profiling

Memory Map...

Inline Assembly...

Function Editor (Open Ini File)...

Project

Target1

Source Group1

simple\_move.asm

Build Output

Build target 'Target 1'

linking...

Program Size: data=8.0 xdata=0 code=14

"a" - 0 Error(s), 0 Warning(s).

Enter or leave a debug session

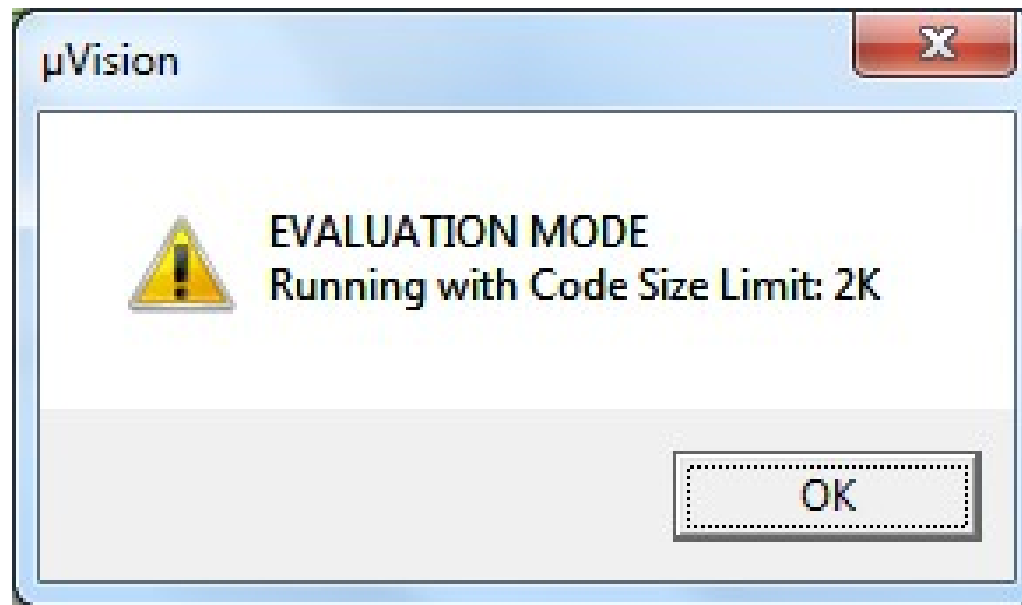
Simulation

CAP NUM SCR

After a successful build of the project, following are the resulting files

1. Binary code is placed in **filename.obj** for each source file in a project.
2. All these obj files are linked into a single binary named using the project name without any extension.
3. If enabled by user <<**filename.hex**>> file is created for the project.

The user can now start the Debug session to check execution of the program.



A popup of Evaluation version is presented which shows the limitation of the mode.

On clicking "OK" the popup closes and the user is presented with the **Debug Mode** user interface (UI)

Opcodes are binary or hexadecimal codes that represent the instructions a microcontroller or processor executes. These are part of machine language and directly control the hardware.

## User interface in Debug mode

The screenshot shows the Keil uVision IDE interface during a debug session. The main window displays assembly code for a program named 'simple\_move.asm'. The assembly code is as follows:

```
10:      mov @r1,a
:0x0007  F7      MOV      @R1,A
11:      inc r0
:0x0008  08      INC      R0
12:      inc r1
13:
```

The 'Registers' window on the left shows the state of the processor registers. The 'Command' window at the bottom left shows the command 'Load "E:\\ACADEMIC\\PG\\RA\\pt-51\\programs\\a"'. The 'Memory' window at the bottom right shows the memory contents starting at address 0x0000.

**Register window**

Register	Value
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
sp	0x07
sp_max	0x07
PC	0x0000
auxr1	0x00
dptr	0x0000
states	0
sec	0.00000000
psw	0x00

**Disassembly window** – shows the Opcodes for the Program loaded in the debugger.

**User program window** – shows the Assembly or High Level Program loaded in the debugger.

**Command window**

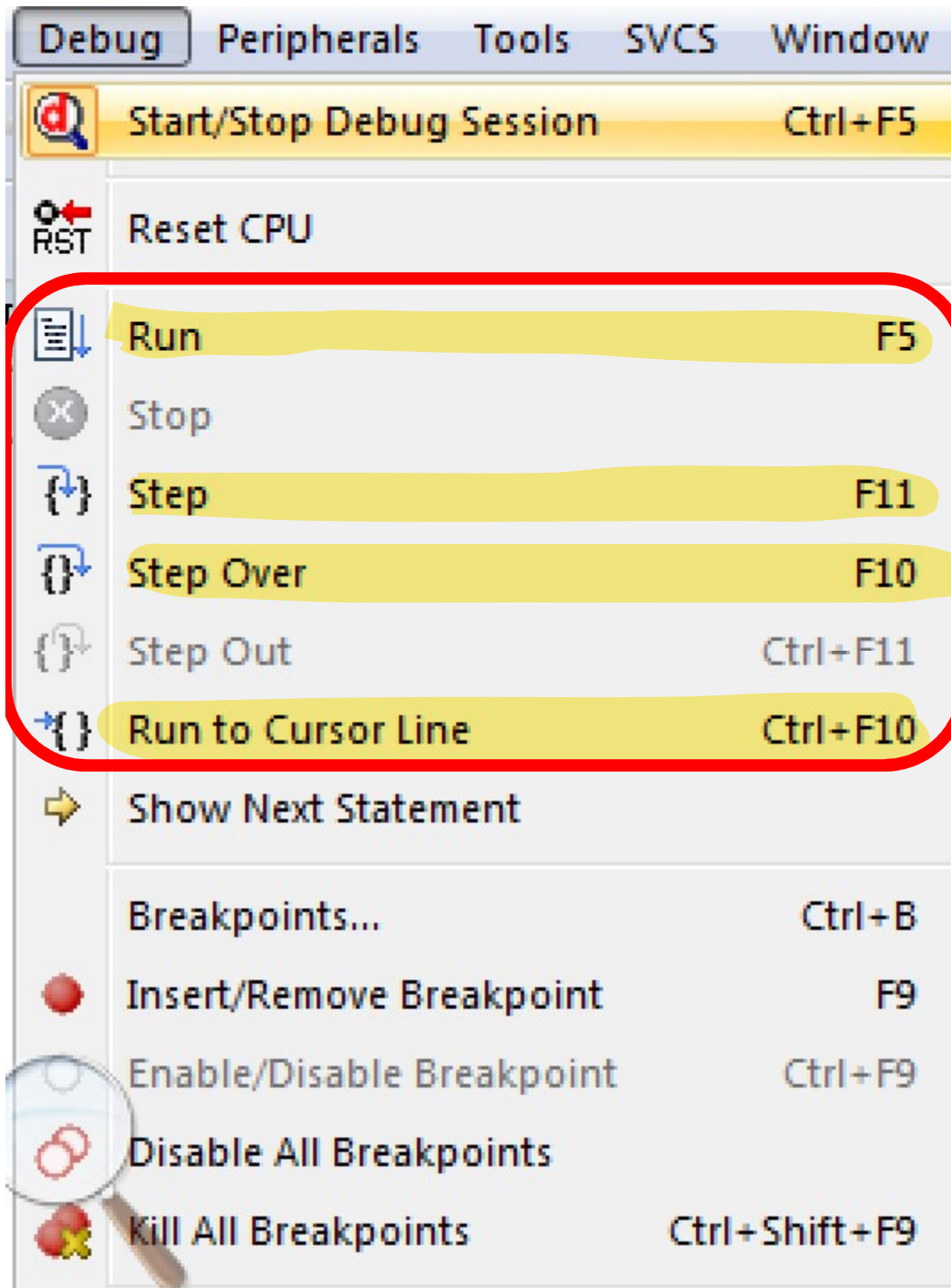
**Memory window**

Address	Value
D:0x00	00 00
D:0x18	00 00
D:0x30	00 00
D:0x48	00 00
D:0x60	00 00
D:0x78	00 00
D:0x90	FF 00

ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess  
Modify memory at cursor location

Simulation t1: 0.00000000 sec CAP NUM SCRL OVR R/W





## User can execute instructions in multiple modes :

1. **Run (F5)** – Continues executing the program until the next active breakpoint is reached or till the program termination.
2. **Step (F11)**-- Executes a single-step into a function; Executes the current instruction line.
3. **Step Over (F10)** – Executes a single-step over a function.
4. **Run to Cursor Line (Ctrl+F10)**  
Allows user to place a cursor and run the program till that line.

# Details of Disassembly and Memory window

Disassembly

```
10: CLR P1.4
C:0x0100 C294 CLR 0x90.4
11: MOV R0, #31D
12: START:
C:0x0102 781F MOV R0, #0x1F
13: SETB EA
C:0x0106 D2AF SETB ETO(0xA8.1)
15: MOV TMOD, #0001B ; Sets Timer 0 to MODE1 (16 bit timer). Timer 1 is not used
C:0x0108 7589 MOV TMOD(0x89), #0x01
16: MOV TH0, #00H ; Loads TH0 register with FCH
C:0x010B 758C00 MOV TH0(0x8C), #0x00
17: MOV TLO, #00H ; Loads TLO register with 18H
```

Disassembly window

Address

Instruction mnemonic in assembly language

Opcode

Memory 1

Address: C:0x100 Enter address here

Memory window

C:0x0100	C2	94	78	1F	D2	AF	D2	A9	75	89	01	75	8C	00	75	8A	00	D2	8C	80	FE	C2	8C	C2
C:0x0118	8D	D8	04	B2	94	78	1F	D2	8C	22	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0148	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C:0x0160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Data at address 0x0100

Call Stack + Locals Memory 1

# Code and Data memory access

The image displays two screenshots of a memory viewer interface, likely from a debugger, illustrating code and data memory access.

**Top Screenshot:**

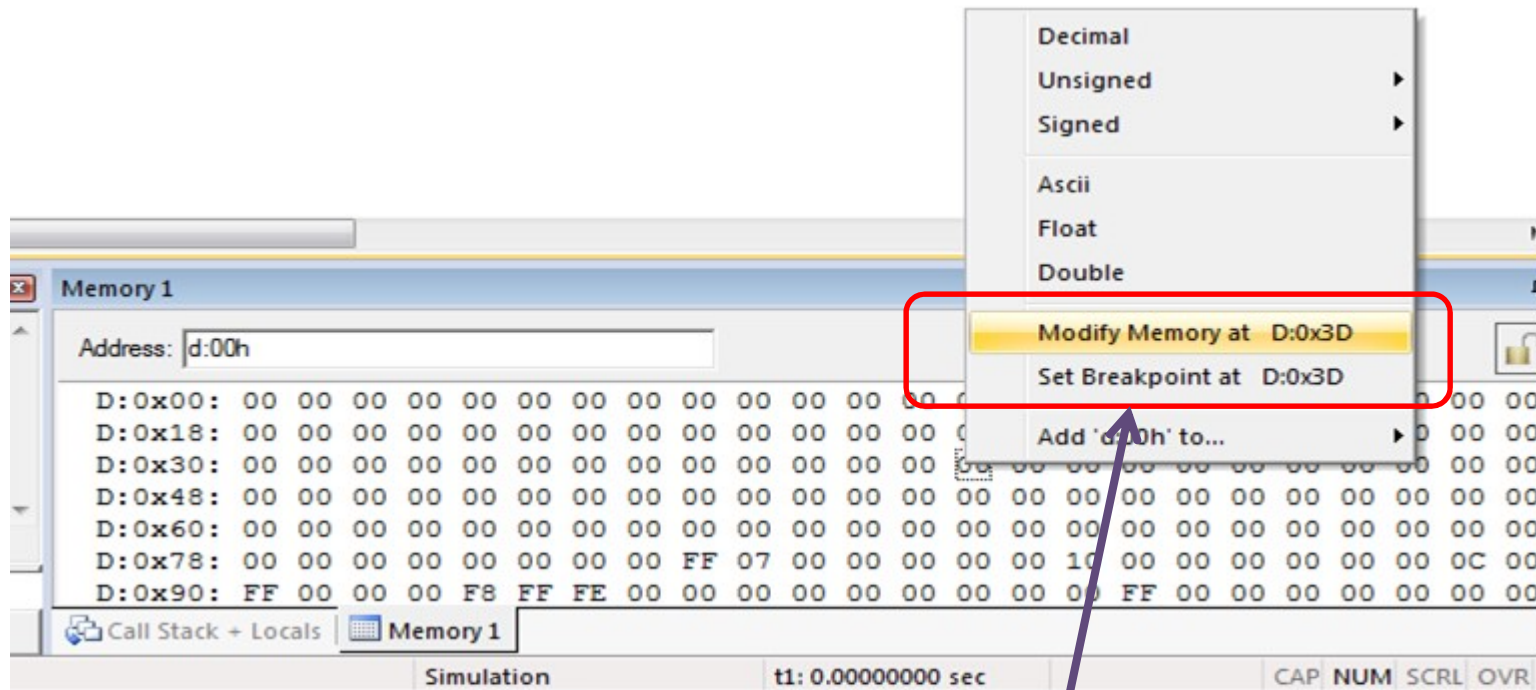
- Address:** `d:00h` (highlighted with a red box).
- Annotation:** *d: refers to data memory* (highlighted with a red box).
- Memory View:** Shows data memory starting at address `D:0x00`. The values are mostly zeros, with some non-zero values at higher addresses: `D:0x78` contains `FF 07 00 00 00 00 10 00 00 00 00 0C 00` and `D:0x90` contains `FF 00 00 00 F8 FF FE 00 00 00 00 00 00 FF 00 00 00 00 00`.

**Bottom Screenshot:**

- Address:** `c:00h` (highlighted with a red box).
- Annotation:** *c: refers to code segment of the memory* (highlighted with a red box).
- Memory View:** Shows code memory starting at address `C:0x0000`. The values are mostly zeros, with some non-zero values at higher addresses: `C:0x0000` contains `78 30 79 40 7A 05 E6 F7 08 09 DA FA 80 FE 00 00 00 00 00 00 00 00 00 00` and `C:0x0090` contains `00 00`.

Both screenshots show a status bar at the bottom with the following information:

- Simulation
- t1: 0.00000000 sec
- L:11 C:9 (top) / L:10 C:14 (bottom)
- CAP NUM SCRL OVR R/W



During execution, user can right click on the required memory location in the memory window to modify RAM data. Functionality for selecting the number system in which the memory contents are to be displayed is also available .

**Note:** To initialize memory contents on hardware, user has to add necessary instructions in the program code.

Registers	
Register	Value
[-] Regs	
r0	0x30
r1	0x40
r2	0x05
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
[-] Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x0000
auxr1	0x00
[-] dptr	0x0000
[0]	0x0000
[1]	0x0000
states	0
sec	0.00000000
[-] psw	0x00
p	0
f1	0
ov	0
rs	0
f0	0
ac	0
cy	0

The Registers window provides access to all the registers including the flag register , DPTRs etc.



E:\ACADEMIC\PG\RA\pt-51\programs\a.uvproj - µVision4

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Registers

Register

Reg

r0

r1

r2

r3

r4

r5

r6

r7

Sys

a

b

sp

sp\_

PC

aux

dptr

state

sec

psw

Project

Command

Running

Load "E:\\ACADEMIC\\PG\\RA\\pt-51\\programs\\a"

ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess

Show or hide the Memory 2 Window

View

Status Bar

Toolbars

Project Window

Books Window

Functions Window

Templates Window

Source Browser Window

Build Output Window

Find In Files Window

Command Window

Disassembly Window

Symbol Window

Registers Window

Call Stack Window

Watch Windows

Memory Windows

Serial Windows

Analysis Windows

Trace

System Viewer

Toolbox Window

Periodic Window Update

Assembly

```
5: mov r0,#src
0000 7830 MOV R0,#0x30
6: mov r1,#dst
0002 7940 MOV R1,#0x40
7: mov r2,#cnt
8:
```

simple\_move.asm

```
src equ 30h
dst equ 40h
cnt equ 5

mov r0,#src
mov r1,#dst
mov r2,#cnt

djnz r2,back
```

Memory 1

Address: d:00h

D:0x00:	35	45	00	00	00	00	00	00
D:0x18:	00	00	00	00	00	00	00	00
D:0x30:	00	00	00	00	00	00	00	00
D:0x48:	00	00	00	00	00	00	00	00
D:0x60:	00	00	00	00	00	00	00	00
D:0x78:	00	00	00	00	00	00	00	00
D:0x90:	FF	00	00	00	F8	FF	FE	00

Call Stack + Locals

Memory 1

Simulation

If some windows are not being displayed then use the "View" menu to get them on the window.

r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x00...
auxr1	0x00
dp1r	0x0000
states	0
sec	0.0000...
psw	0x00

```
simple_move.asm
1  src equ 30h
2  dst equ 40h
3  cnt equ 5
4
5  mov r0, #src
6  mov r1, #dst
7  mov r2, #cnt
8
9  back:  mov a, @r0
10        mov @r1, a
11        inc r0
12        inc r1
13
14        djnz r2, back
15
16  loop:  sjmp 1
17
18  end
```

To set a breakpoint, user can click in the marked area against the corresponding line of code

The screenshot shows a debugger interface with two main panes. The left pane, titled 'Registers', displays a list of registers and their values. The right pane, titled 'Disassembly', shows the assembly code being executed. A red dot is placed on the left margin of the disassembly window, specifically on line 12, indicating a breakpoint. A red box highlights the left margin of the disassembly window. A blue arrow points from the text box to the red dot. Another blue arrow points from the text box to the disassembly line 12.

Register	Value
Regs	
r0	0x30
r1	0x40
r2	0x05
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x00...
auxr1	0x00
dpnr	0x0000
states	0
sec	0.0000...
psw	0x00

```
11:          inc r0
C:0x0008    08          INC          R0
12:          inc r1
13:
C:0x0009    09          INC          R1

simple_move.asm
1  src equ 30h
2  dst equ 40h
3  cnt equ 5
4
5  mov r0,#src
6  mov r1,#dst
7  mov r2,#cnt
8
9  back:  mov a,@r
10         mov @r1,a
11         inc r0
12         inc r1
13
14  djnz r2,back
15
16  loop:  sjmp loop
17
18  end
```

The breakpoint is shown as a red dot against the line.

The breakpoint is automatically displayed at the equivalent line in the disassembly window too.



# Peripherals menu

The screenshot shows a microcontroller IDE with the 'Peripherals' menu open. The menu options are: Interrupt, I/O-Ports, Serial, Timer, TWI, SPI, Keyboard Interrupt, EEPROM Data, Flash Memory, and Clock Control. The 'I/O-Ports' sub-menu is expanded, showing Port 0, Port 1, Port 2, Port 3, and Port 4. A text box on the right states: 'Various Peripherals can be accessed through the "Peripherals" menu.'

The 'Registers' window on the left shows the following registers and values:

Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC \$	C:0x0000
auxr1	0x00
dptr	0x0000
states	0
sec	0.00000000
psw	0x00

The assembly code window shows the following code:

```
1 src equ 30h
2 dst equ 40h
3 cnt equ 5
4
5 mov r0,#src
6 mov r1,#dst
7 mov r2,#cnt
8
9 back: mov a,@r0
10      mov @r1,a
11      inc r0
12      inc r1
13
14 djnz r2,back
```

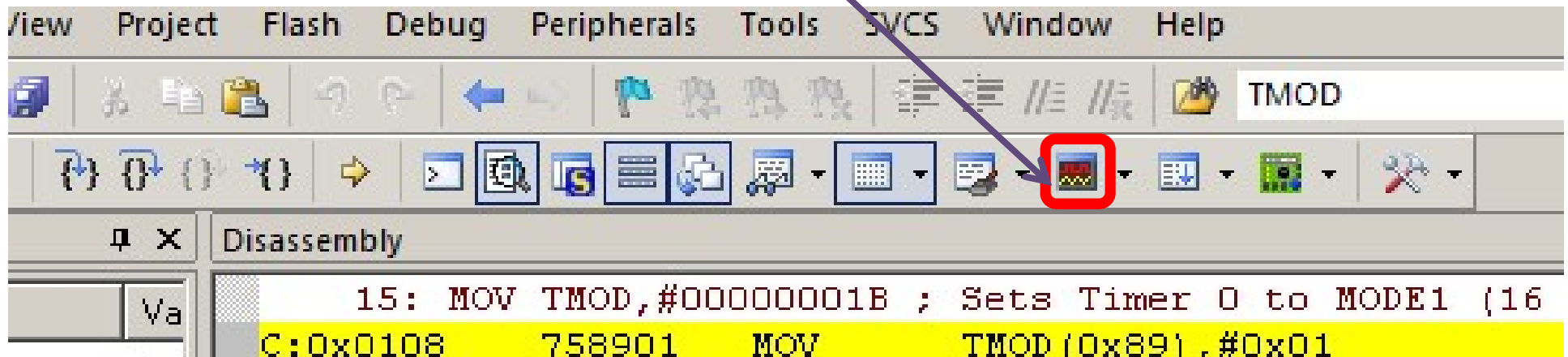
The 'Timer/Counter 0' window is open, showing the following settings:

- Mode: 0: 13 Bit Timer/Counter
- Timer: Timer
- TCON: 0x00, TMOD: 0x00
- TH0: 0x00, TL0: 0x00
- ☒ TO Pin, ☐ TFO
- Status: Stop
- ☐ TR0, ☐ GATE, ☒ INTO#

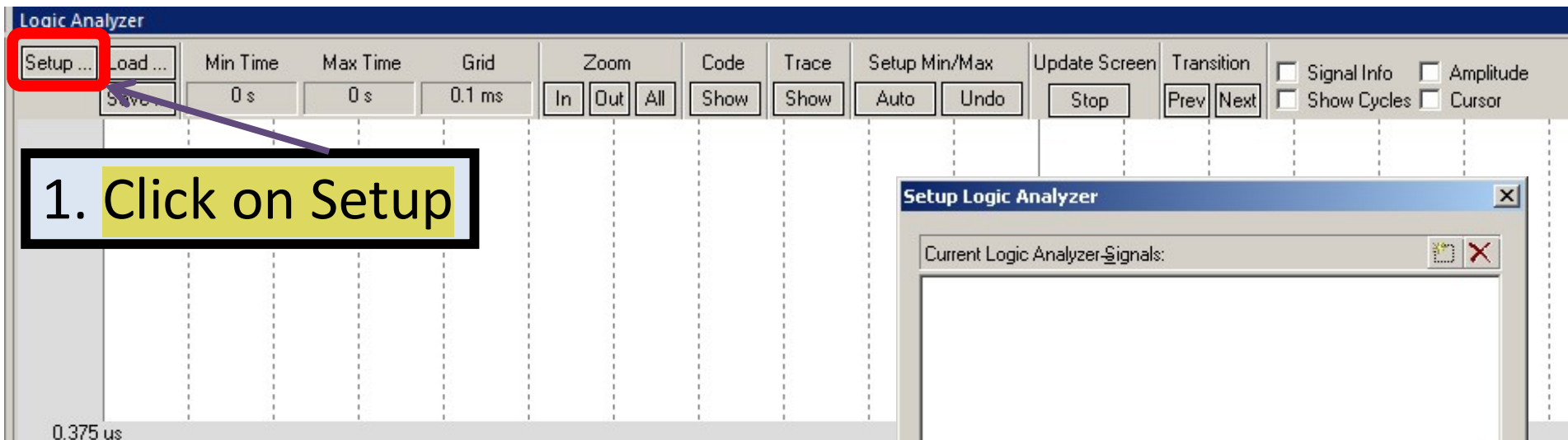
A text box on the right states: 'Window corresponding to Timer 0'.

# Logic analyzer

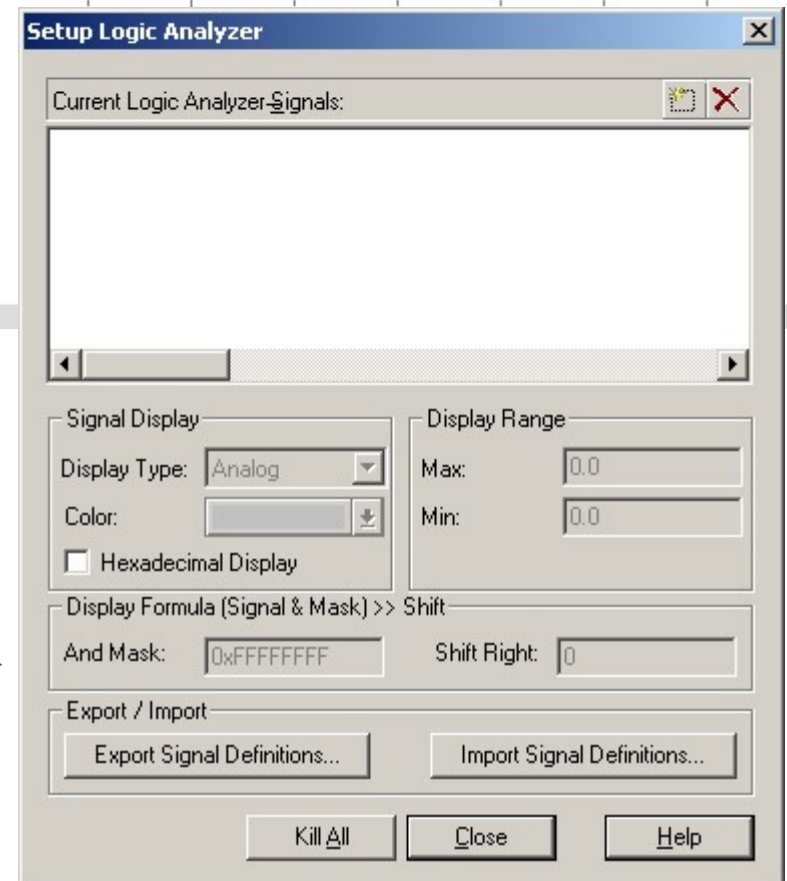
To start the logic analyzer click on the highlighted icon or go to **View > Analysis Window > Logic Analyzer.**



# Logic Analyzer window

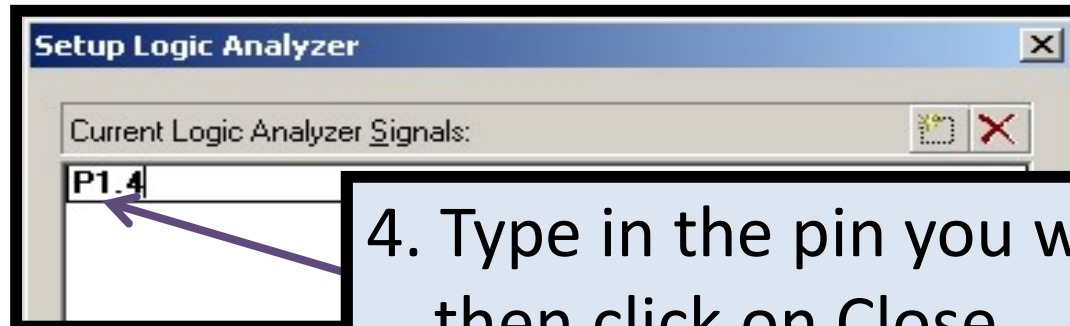


2. The setup window appears.



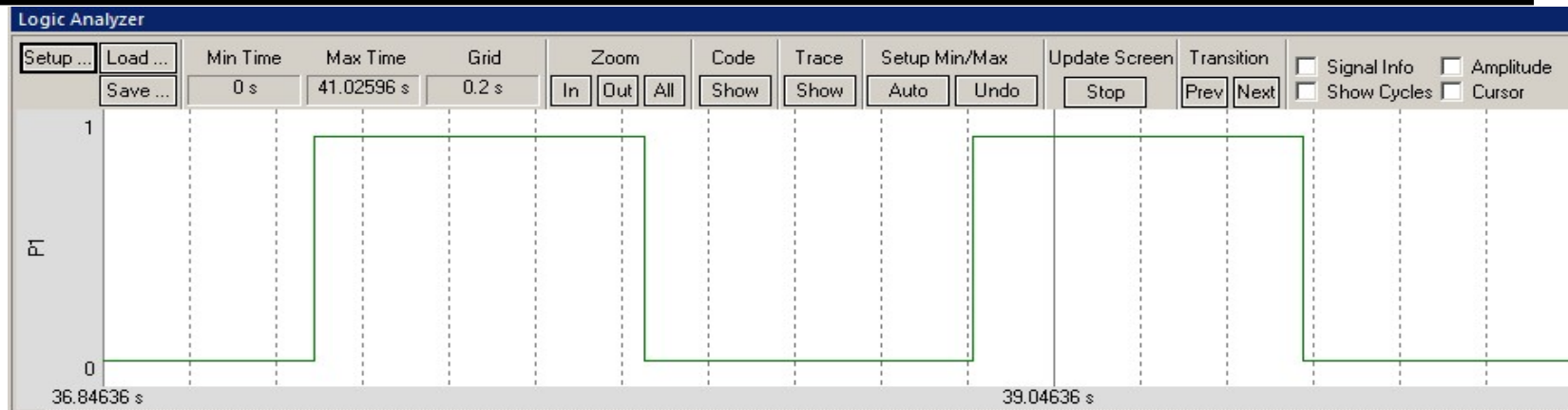


3. Click on the “New/insert” icon.



4. Type in the pin you want to monitor then click on Close.

5. After running a simulation, you can pause it and look at the timing waveforms to debug your code.



**Questions ?**

# Thank you

**WEL Labs, IITB**  
**2016**

For doubts/errors in this PPT contact :  
Suryakant Toraskar e-mail: [smtoraskar.iitbombay@gmail.com](mailto:smtoraskar.iitbombay@gmail.com) location : WEL5