Dr. Ambedkar Institute of Technology, Bengaluru-56

(An Autonomous Institution Affiliated to VTU, Belagavi)
(Accredited by NAAC with Grade "A")

Department of Electronics and Communication Engineering

Microcontroller Lab Manual
(18ECL56)

Prepared By

Triveni.P
Vidyashree C

Student Name: ______________________________

USN: ______________________________

Section: ______________________________

Batch: ______________________________

Vision Statement:

"To excel in education and research in Electronics and Communication Engineering and its related areas through its integrated activities for the society"

Mission Statement:

- To provide high-quality education in Electronics and Communication Engineering discipline and its related areas to meet the growing challenges of the industry and the society through research.
- To be a contributor to technology through value-based quality technical education.
- To equip the students with strong foundations in Electronics and communication engineering.

Program Educational Objectives (PEOs):

PEO1: Graduates will have a solid foundation in electronics and communication engineering.

PEO2: Graduates are technically competent and able to analyze, design, develop and implement electronic and communication systems.

PEO3: Graduates will have sufficient breadth in electronics and its related fields so as to enable them to solve general engineering problems.

PEO4: Graduates are capable of communicating effectively and interact professionally with colleagues, clients, employers and the society.

PEO5: Graduates are capable of engaging in life - long learning and to keep themselves abreast of new developments in their fields of practice.

Program Outcomes (POs):

PO01: Engineering knowledge- Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO02: Problem analysis- Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO03: Design/development of solutions - Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO04: Conduct investigations of complex problems - Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO05: Modern tool usage - Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO06: The engineer and society - Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO07: Environment and sustainability - Understand the impact of professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO08: Ethics - Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO09: Individual and team work - Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication - Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance - Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage project and in multidisciplinary environment.

PO12: Life-long learning- Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1: Capability to use mathematical Techniques to model real-time problems, to optimize the implementation using mathematical techniques and to analyze the system performance.

PSO2: Ability to understand, analyze and apply the Electronic circuits, VLSI circuits, Antennas, Microwave, Microcontrollers, embedded Controllers, and Communication System concepts to design and implement real-time applications.

PSO3: Ability to identify and have the social and ethical responsibilities for the attainment of Society and to become an entrepreneur.

Sub Title : Microcontroller Lab		
Sub Code: 18ECL56	**No. of Credits:1=0 : 0 : 1 (L-T-P)**	**No. of lecture hours/week : 02**
Exam Duration : 3 Hours	**CIE +Assignment + SEE = 50 + 0 + 50 =100**	**Total No. of Contact Hours :26**

Course objectives:

1. To learn the architecture of 8051 Microcontroller.
2. To learn the Instruction set and Embedded C for MCS51.
3. Ability to write a ALP and C program for a given algorithm and implement the same
4. To learn the I/O ports and interfacing techniques with MCS51.
5. Ability to develop single chip solution using MCS51.

Unit No.	Syllabus contents	No of Hours	RBT level
PART-A	**PROGRAMMING WITH 8051 MICROCONTROLLER**		
1.	**Data Transfer**: Block move, Block Exchange, Finding largest/Smallest element in an array, sorting.	2	L1,L2,L3
2.	**Arithmetic Instructions**: Addition/subtraction, multiplication and division, square, Cube	3	L1,L2,L3
3.	**Counters: U**p/down, BCD counter	2	L1,L2,L3
4.	**Boolean & Logical Instructions (Bit/Byte):** Logic gates, Adder/Subtractor, multiplexer circuits, Palindrome	2	L1,L2,L3
5.	**Code conversion**: BCD – ASCII; Binary - BCD.	2	L1,L2,L3
6.	**Programs to generate time delay using** on-Chip timer/Counter, Program as event counter, Programs using serial port and Programs using interrupts.	2	L1,L2,L3
PART B	**INTERFACING PROGRAMS:**		
1.	Program to display BCD UP counting	2	L1,L2,L3
2.	DAC interfacing(square, triangular and ramp)	2	L1,L2,L3
3.	Program to control the stepper motor	4	L1,L2,L3
4.	Program to display the key pressed	2	L1,L2,L3
5.	LCD interfacing		L1,L2,L3

Course Outcomes:

CO1. Understand the architectural features of microcontrollers.
CO2. Able to know the instruction set of MCS51 and write Assembly and High level language Programs.
CO3. Study the various features of MCS51.
CO4. Study the applications of MCS51 for real time systems.
CO5. Development of single chip solutions

Cos	Mapping with POs
CO1	PO2, PO3

CO2	PO2, PO3
CO3	PO2, PO3, PO11, PO12
CO4	PO2, PO3, PO11, PO12
CO5	PO2, PO3, PO11, PO12

Software used	**Keil µVision 4**
Hardware	**AT89S52 kit**

Text Book:	
1.	**Kenneth J, Ayala**, “The 8051 Microcontroller Architecture, Programming & Applications”, 2edition, 1996 / Thomson Learning 2005.
2.	**Muhammad Ali Mazidi, Janice Gillespie Mazidi and Rollin D. McKinlay,** “The 8051 Microcontroller and Embedded Systems – using assembly and C”,; PHI, 2006 / Pearson, 2006.

Dr. Ambedkar Institute of Technology, Bengaluru

Department of Electronics & Communication Engineering

MCIROCONTROLLER LAB_(ECL46/18ECL56/18ECL57)

Rubrics for evaluation

Each program/ experiment needs to be evaluated for 30 marks based on the following 5 categories (Design with specification, Conduction/Implementation, Analysis/Interpretation of results, Record write up & Viva-voce) on a scale of marks as stated in the following table

Category	Poor	Fair	Good	Excellent	Outstanding	PO's and PSO's
Design with specification (5M)	Students has not yet all understand the statement. (0-1)	Students has understand the statement wrong. (1-2)	Students need a hint to clearly understand the statement (2-3)	Students has understand the statement after theory explanation (3-4)	Students has clearly understand the statement independently (4-5)	PO1,PO2,PO3,PO4 PSO1,PSO2
Conduction and Implementation (8M)	Students has no idea how to use basic tools of the software. (0-1)	Students has full command on the basic tools of the software. (2-3)	The code is several syntax errors. Important parts of code are missing. (4-5)	The code is completely functional and responds correctly need minor help for producing the correct outputs. (5-7)	The code is completely functional and responds correctly producing the correct outputs (7-8)	PO1,PO4,PO5,PO8,PO9 PSO1,PSO2
Analysis/ Interpretation of results (5M)	Not able to get results (0-1)	Partial results (1-2)	Getting result not able to observe (2-3)	Getting result not able justify (3-4)	Getting result and justifying correctly the out put (4-5)	PO1,PO2,PO4,PO10 PSO1,PSO2
Record write up (7M)	Without aim of the experiment written logically incorrect experiment. (0-1)	Written the experiment logically in correct with aim. (1-3)	Written the experiment with aim and with incorrect output. (3-5)	Written the experiment with aim and expected result before and after execution. (5-6)	Written the experiment with aim,code,expected out put before execution and after, with required calculation and block diagrams and wave forms (6-7)	PO1,PO8,PO9,PO10 PSO1,PSO2
Viva-voce (5M)	Answered only 1 question with a hint. (0-1)	Answered all 3 questions with some hint (1-2)	Answered all 4 questions with some hint (2-3)	Answered all 5 questions with some hint (3-4)	Answered all 5 questions without any hint (4-5)	PO1,PO6,PO8,PO9,PO10 PSO1,PSO2

Note: In each column evaluation criteria statements with marks to be mentioned.

Faculty Signature

INDEX

	to its equivalent two digit DECIMAL/8bit BCD number.
8(a)	WALP to convert an 8 bit BCD/2 digit DECIMAL number to its equivalent ASCII number.
8(b)	WALP to convert the given two bytes of ASCII number to its equivalent 8 bit BCD/2 digit DCIMAL number.
9	WALP to generate the output value AAH and 55H alternatively at the Port 0 for every 2 sec.
10	WALP to transfer the string of data using serial communication by configuring the serial port in mode_1 with a baud rate of 9600,1 step bit and 8 data bits.
11	WALP to count the number of inputs to the counter_0 by configuring the timer as counter in mode_1 and display the count in ports.
12	Let X,Y,Z refers to the contents of the memory location 30h,31h,and 32h respectively write an ALP to perform the following logical operations; If X=00 perform the operation Y OR Z. If X=01 perform the operation Y AND Z. If X=02 perform the operation Y XOR Z.
13	WALP to compare the bytes of data present at the memory location 21h and 22h and represent the result of comparison through the bits whose addresses are 00h and 01h. If (21h)<(22h)then clear the bit at 01h and also set the bit at 00h. If (21h)>(22h)then set the bit at 01h and also clear the bit at 00h. If (21h)=(22h)then set the bit at 01h and also set the bit at 00h.
14	Let X,Y AND Z refers to the bits of the bit addressable memory whose addresses 00H,08H and 0FH respectively write an ALP to perform the following logical operation : If X=0 perform If X=0 perform the operation Y OR Z. If X=0 perform If X=0 perform the operation Y AND Z.
15	WALP to stimulate the Boolean expression Y=A+BC
16(a)	WALP to implement an 8 bit binary/2 digits hex up counter. The counter has to count from 00H to FFH.
169b)	WALP to implement an 8 bit binary/2 digits hex down counter. The counter has to count from FFH to 00H.
17(a)	WALP to implement an 2 digits decimal/8 bit BCD up counter. The counter has to count from 00 to 99.
17(b)	WALP to implement an 2 digits decimal/8 bit BCD down counter. The counter has to count from 99 to 00.
	PART-B
1	Write a C-program to accept the input from port 0 and sent to the port 2.
2	Write a C-program to display bcd up counting.
3(a)	Write a C-program to generate triangular waveform using dac
3(b)	Write a C-program to generate square waveform using DAC

4	Write a C-program to display the key pressed
5(a)	Write a C-program to control the stepper motor
5(b)	Write a C-program to control the stepper motor for the required no of rotations.
6	Write a C-program on LCD display

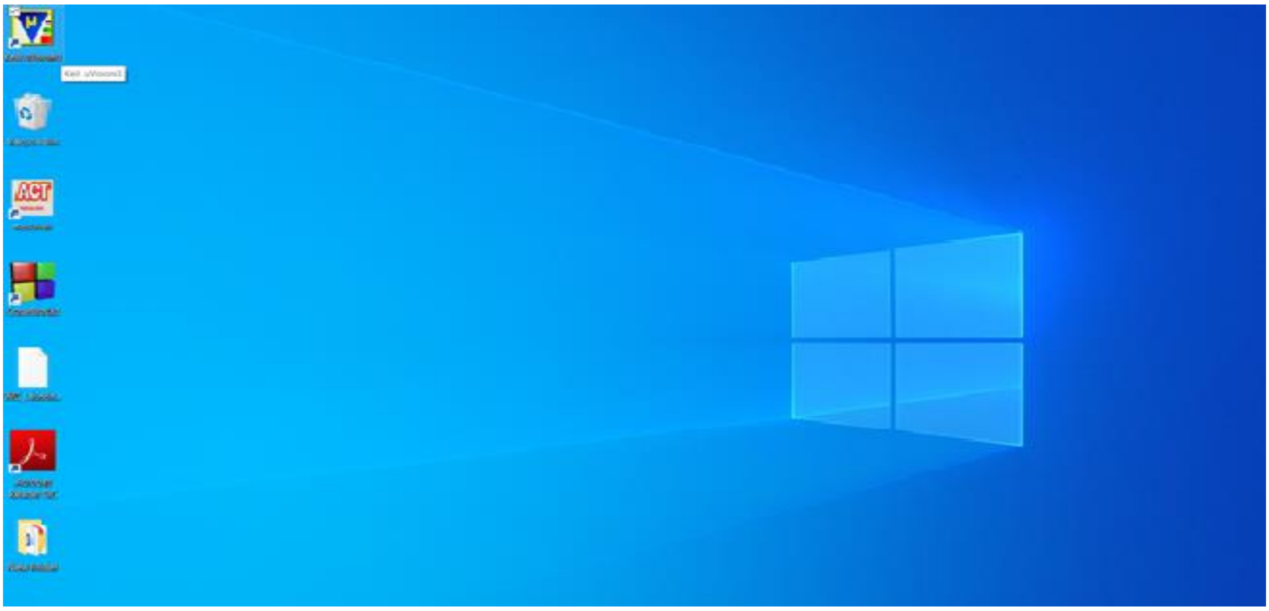
PART-A

ASSEMBLY PROGRAMMING

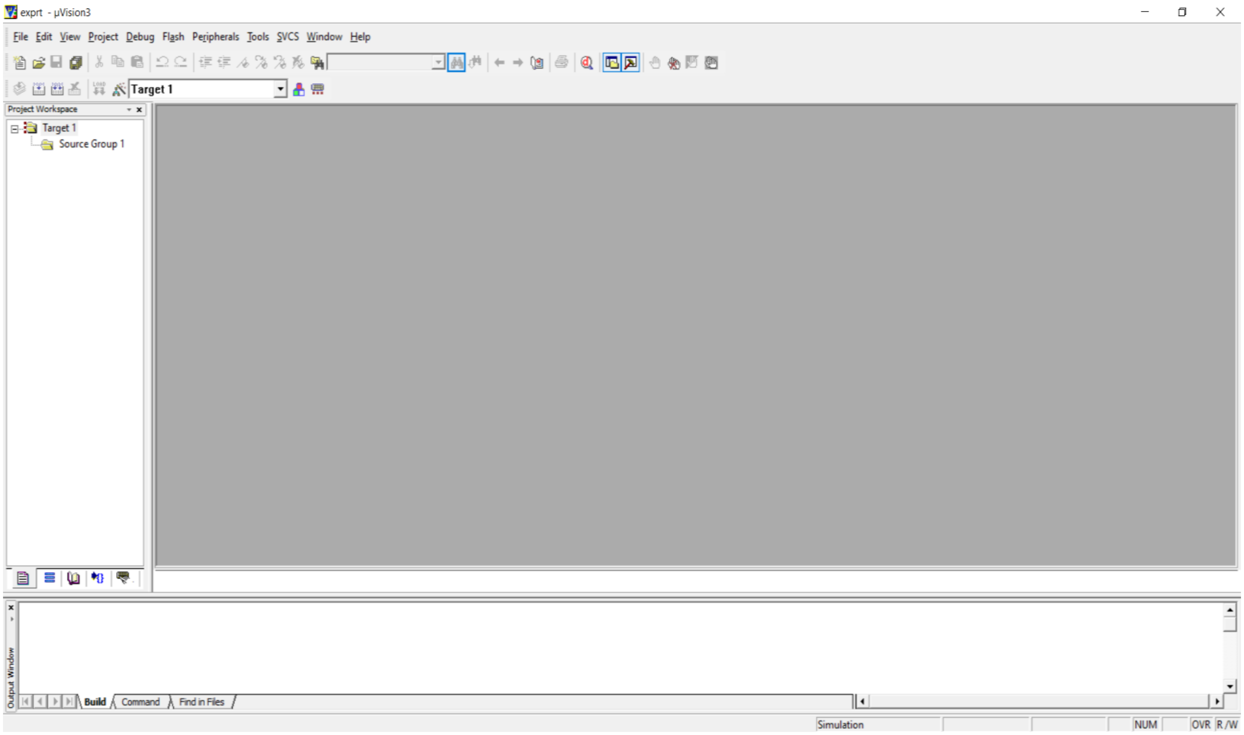
FOLLOW THE STEPS TO EXECUTE

THE MCS51 PROGRAMS USING KEIL SOFTWARE VERSION.3

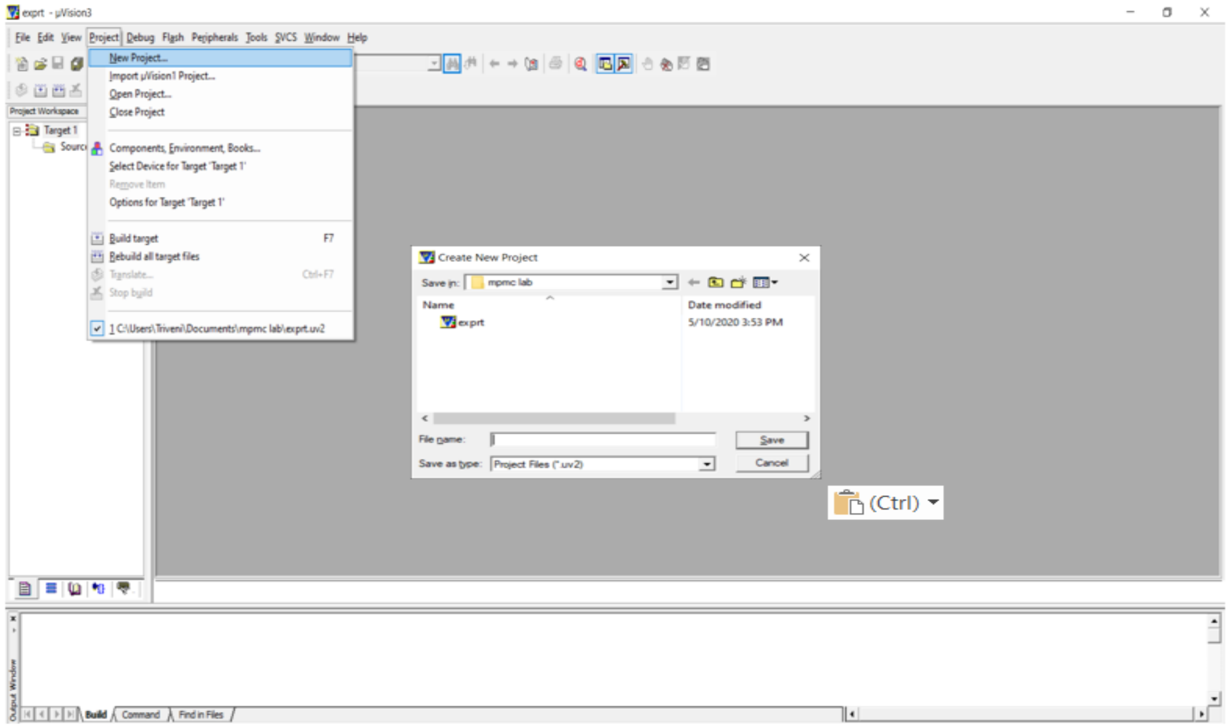
STEP 1: Double click on your keil software



STEP 2: You will get a µVision software work space



STEP 3: Creat your **New project** in your own folder with any name

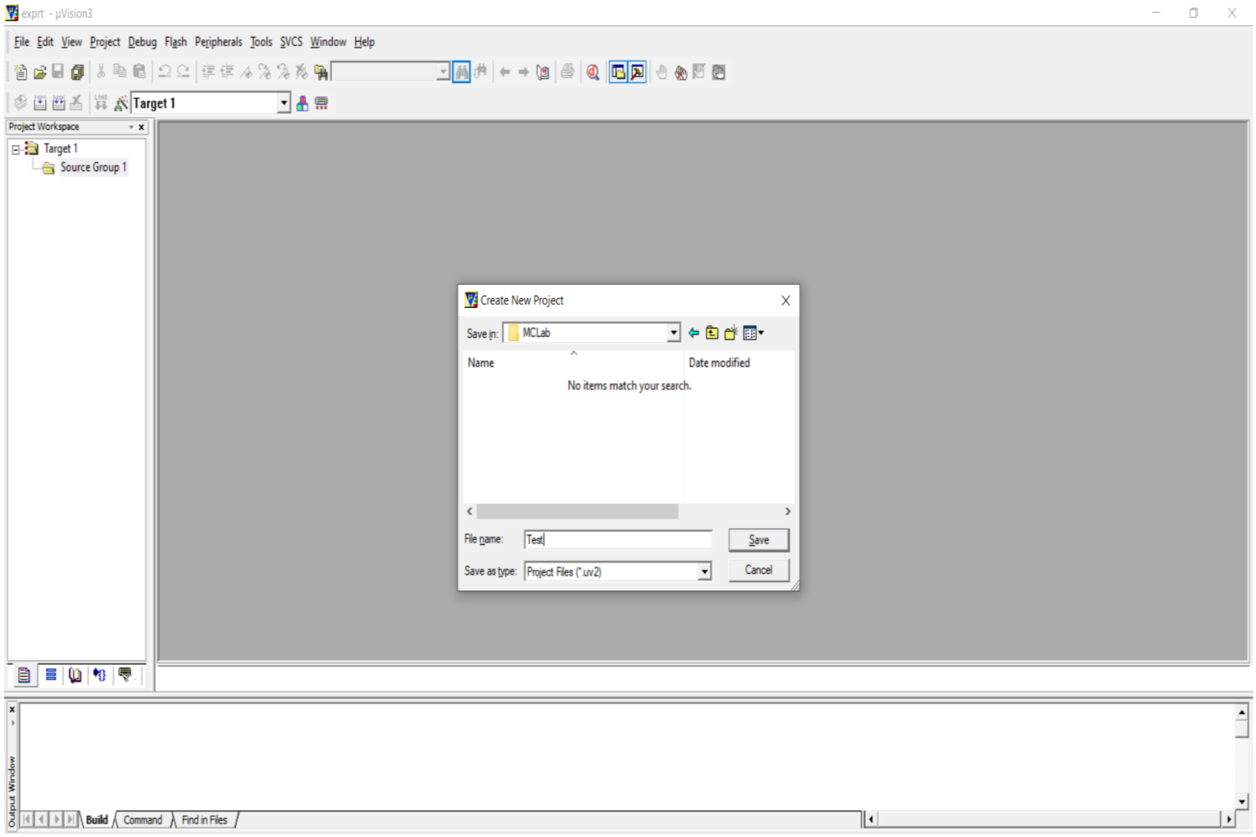


Save in: mpmc lab

exprt 5/10/2020 3:53 PM

File name:

Save as type: Project Files (*.uv2)



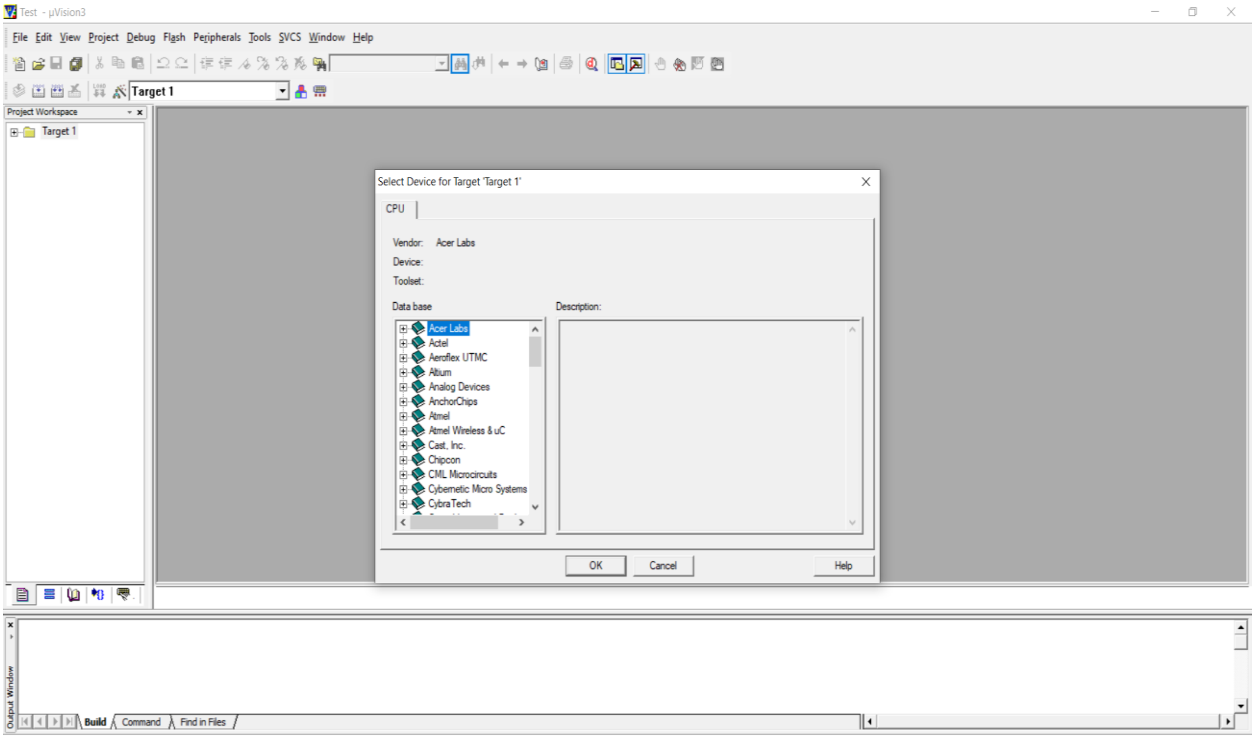
Save in: MCLab

No items match your search.

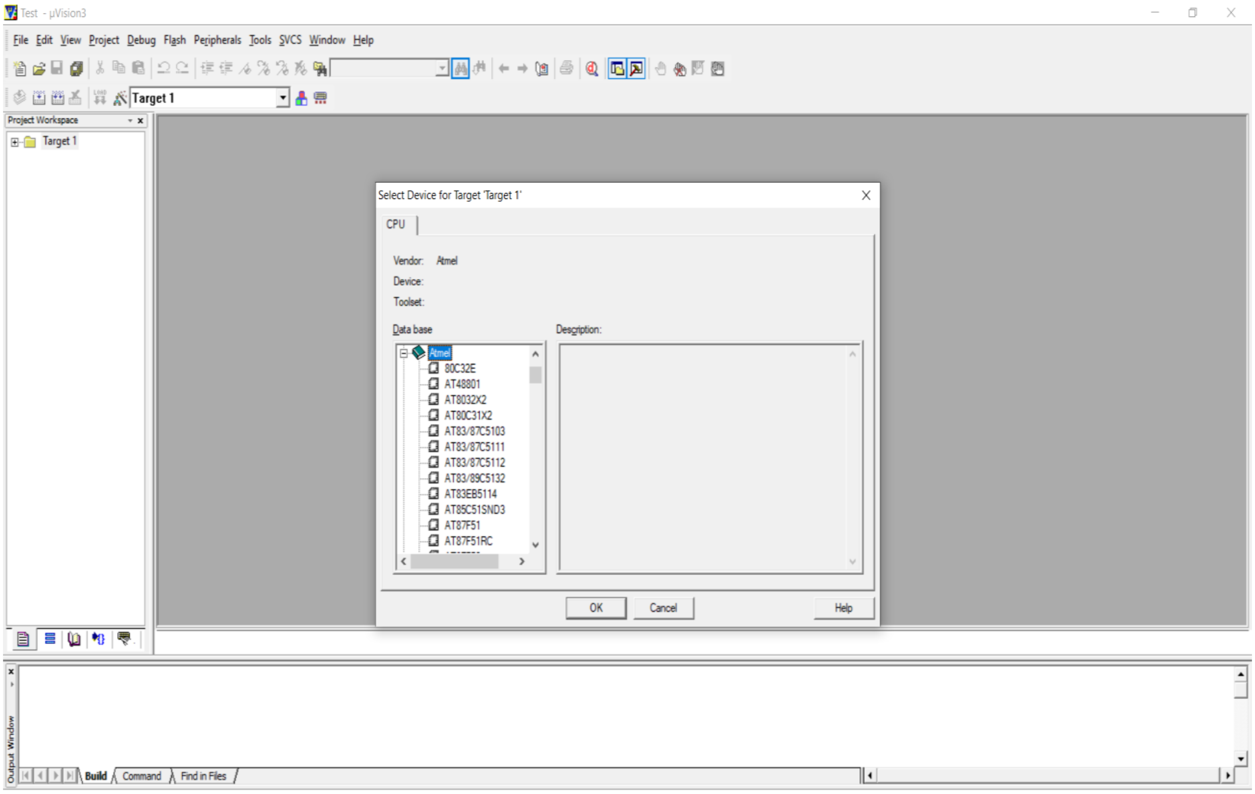
File name: Test

Save as type: Project Files (*.uv2)

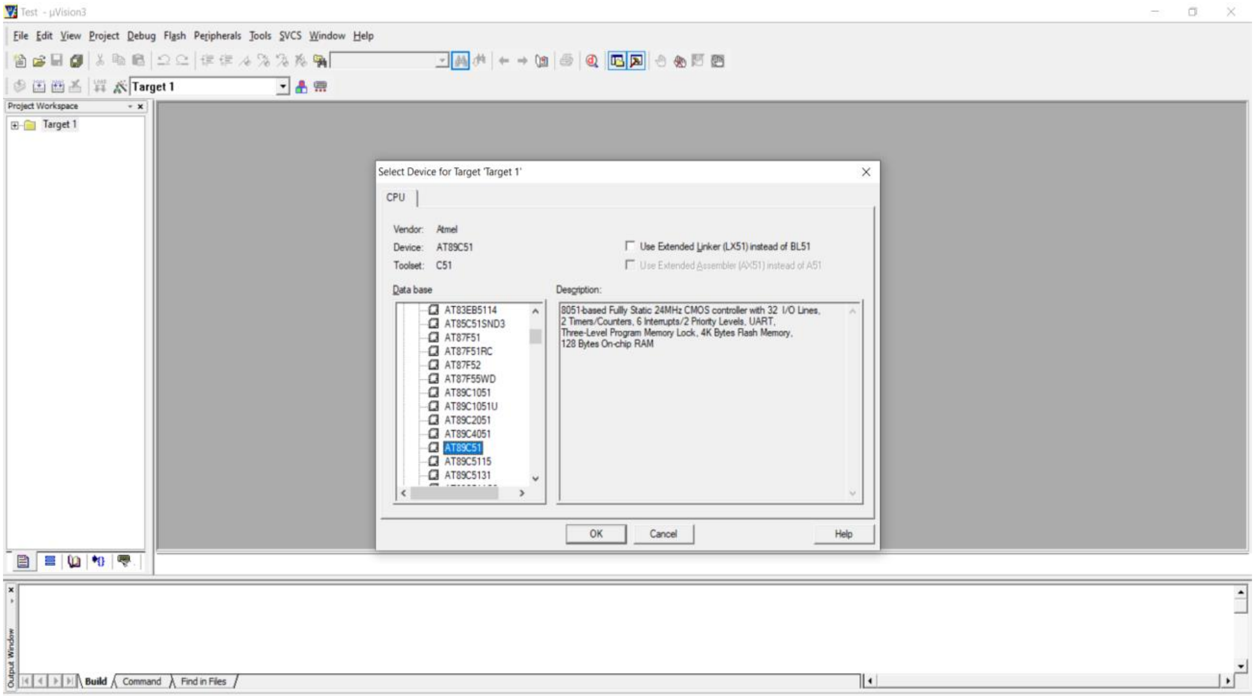
STEP 3: Select your microcontroller family i.e., **Atmel** in that choose your 8051 microcontroller IC number **AT89C51** then say **OK**



Vendor: Acer Labs
Device:
Toolset:



Vendor: Atmel
Device:
Toolset:



Select Device for Target 'Target 1'

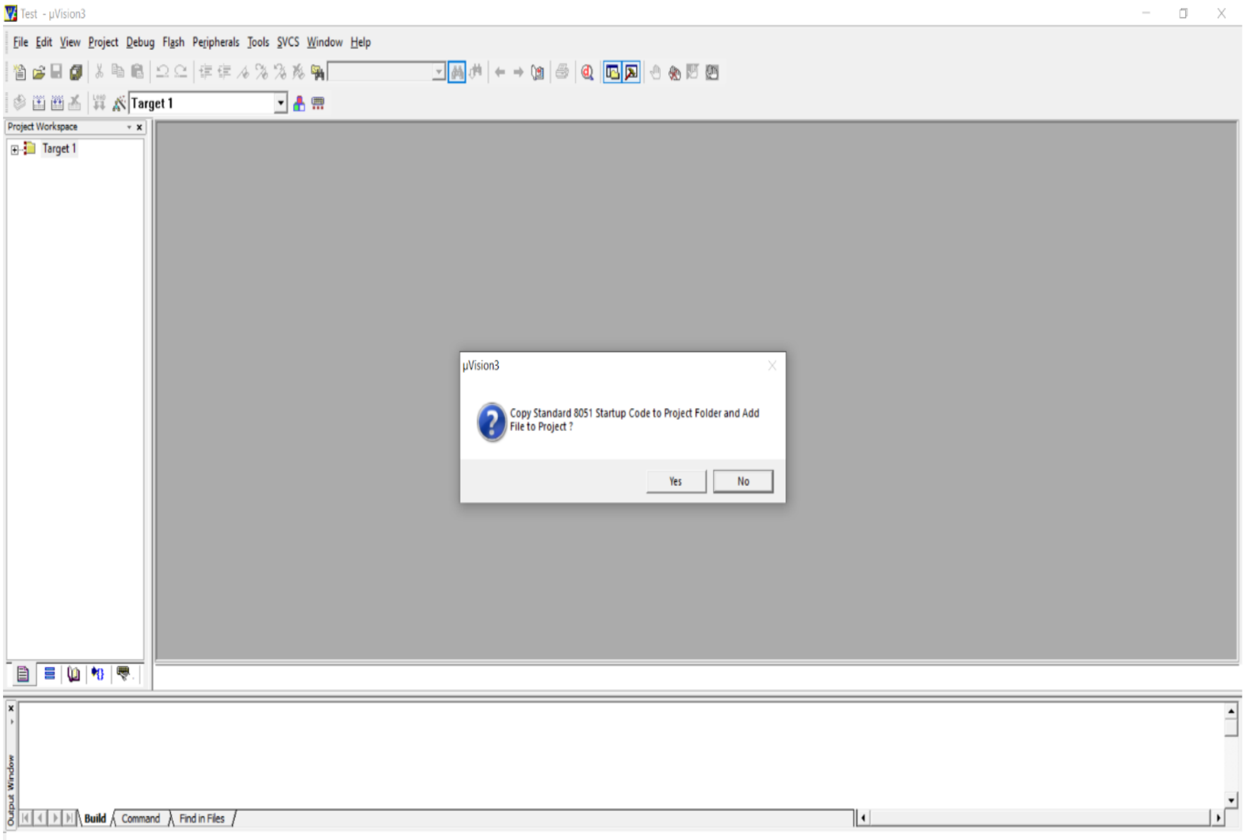
Vendor: Atmel
Device: AT89C51
Toolset: C51

□ Use Extended Linker (LX51) instead of BL51
□ Use Extended Assembler (AX51) instead of A51

Data base: AT83EB5114, AT85C51SND3, AT87F51, AT87F51RC, AT87F52, AT87F55WD, AT89C1051, AT89C1051U, AT89C2051, AT89C4051, AT89C51, AT89C5115, AT89C5131

Description:
8051-based Fully Static 24MHz CMOS controller with 32 I/O Lines, 2 Timers/Counters, 6 Interrupts/2 Priority Levels, UART, Three-Level Program Memory Lock, 4K Bytes Flash Memory, 128 Bytes On-chip RAM

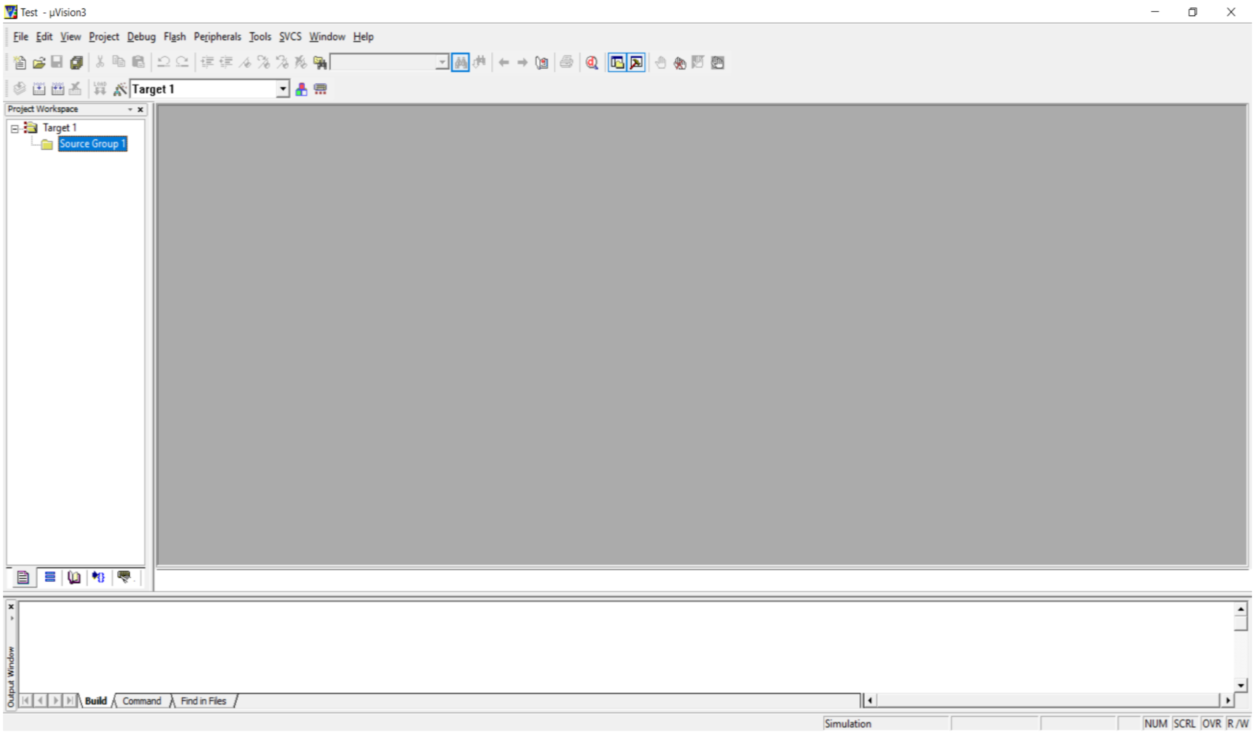
STEP 4: It will ask you whether the startup code is required then better we must select **NO** as we are executing simple academic level programs.

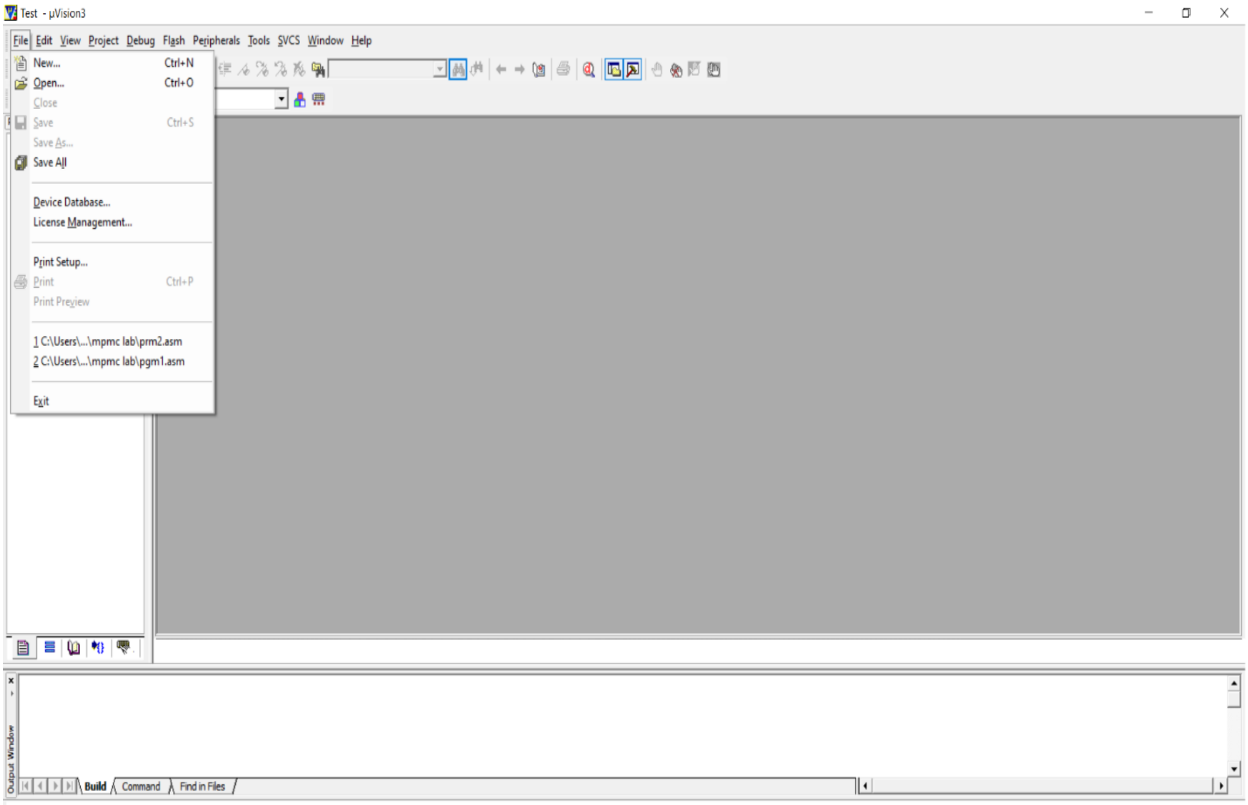


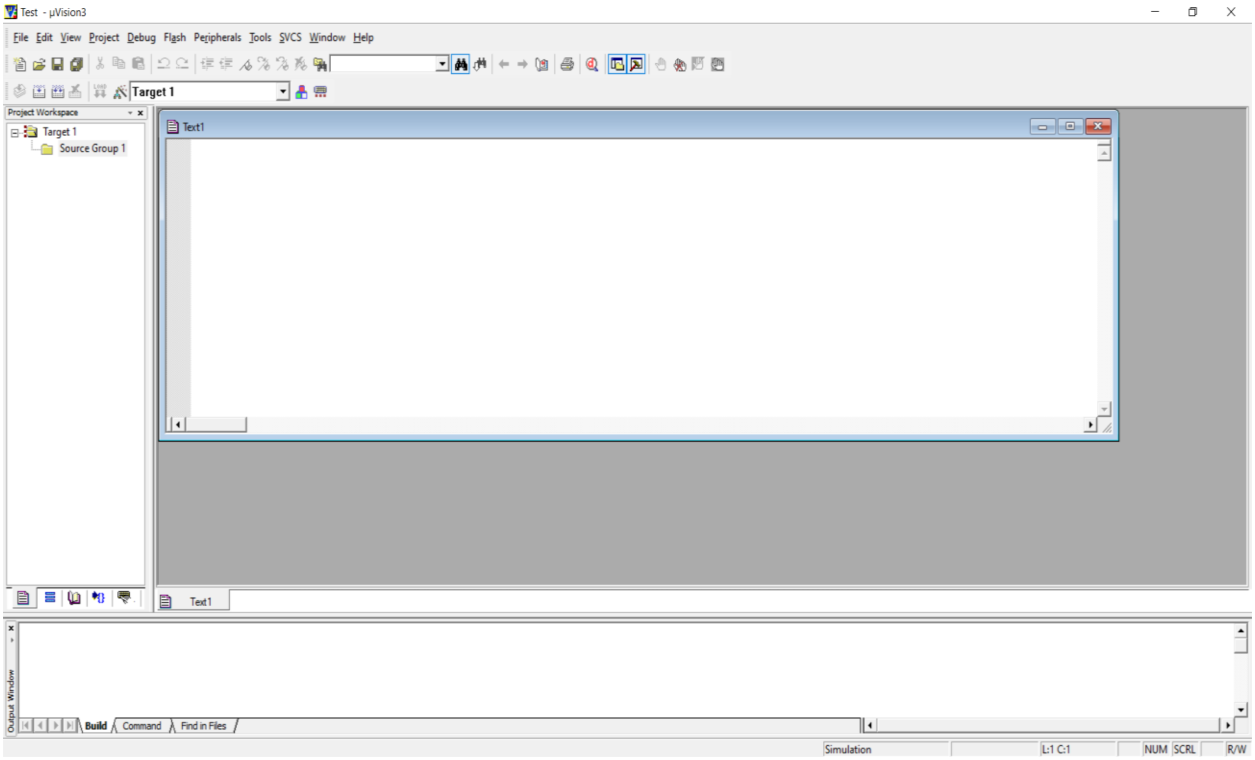
Copy Standard 8051 Startup Code to Project Folder and Add File to Project ?

S

TEP 5: Ones your Target has created successfully then write your program by selecting **New file** from **FILE menu.**

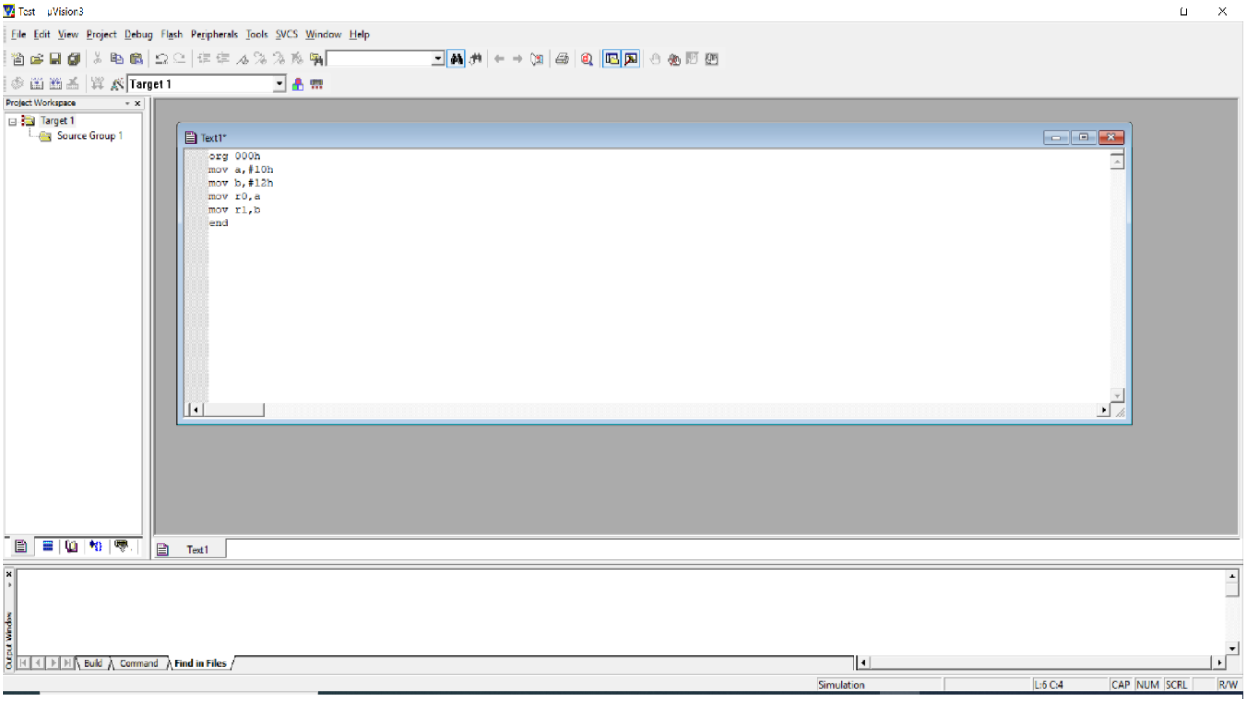




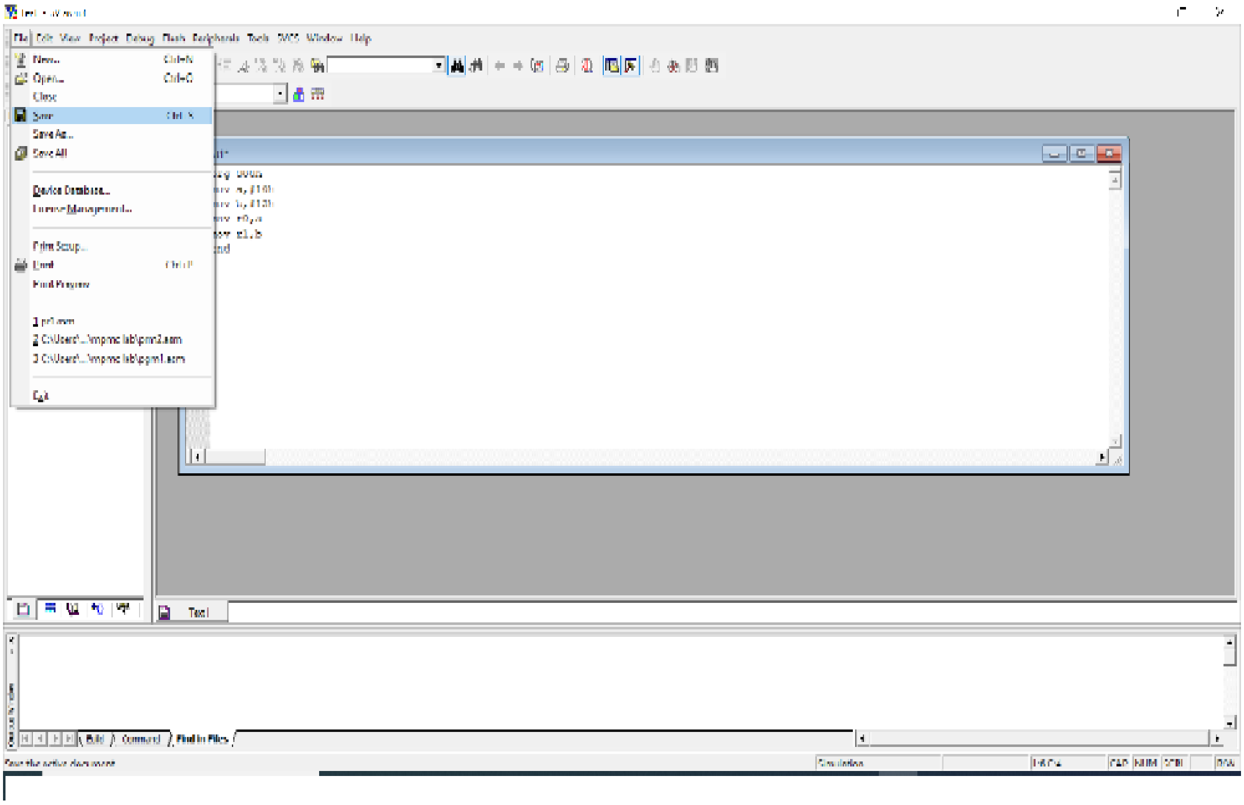


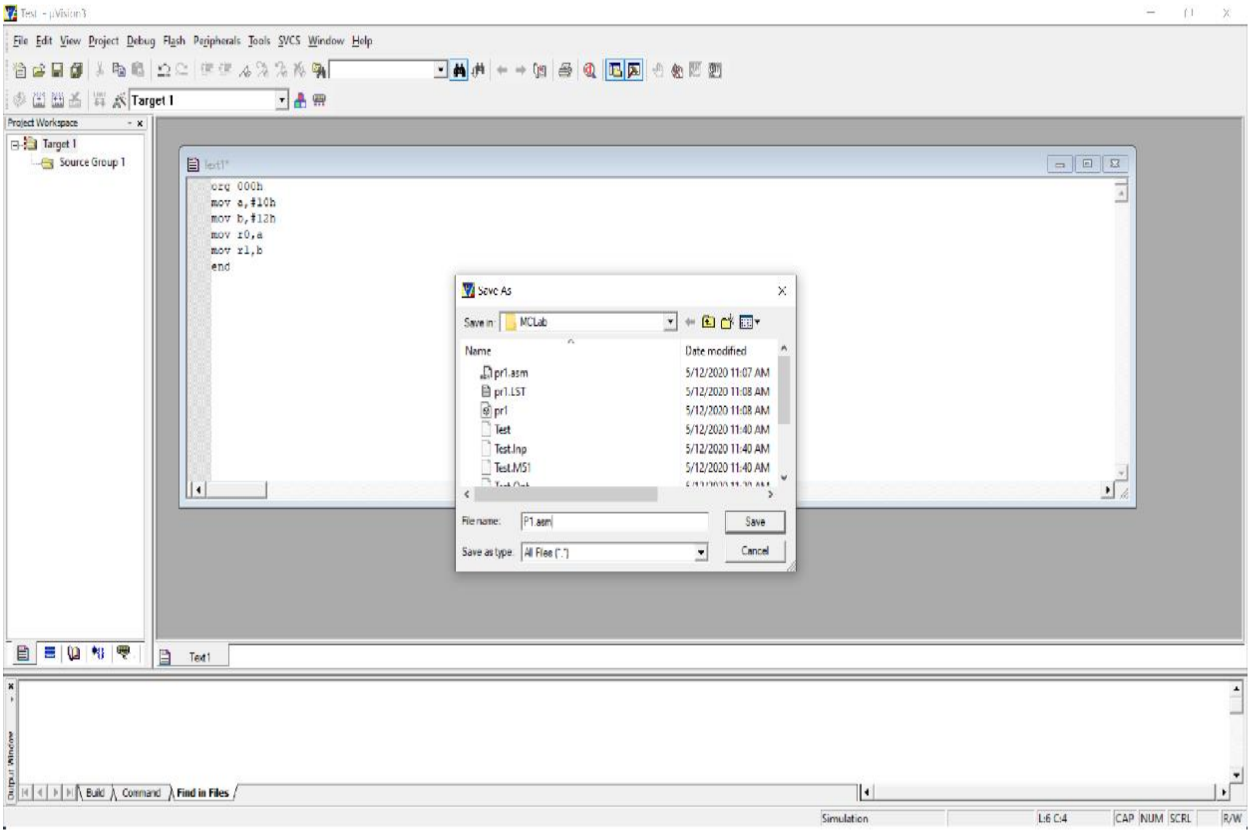
STEP 6: Write your program and save your text file with an extension .asm in your own folder.

Example: xyz.asm



```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```





```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

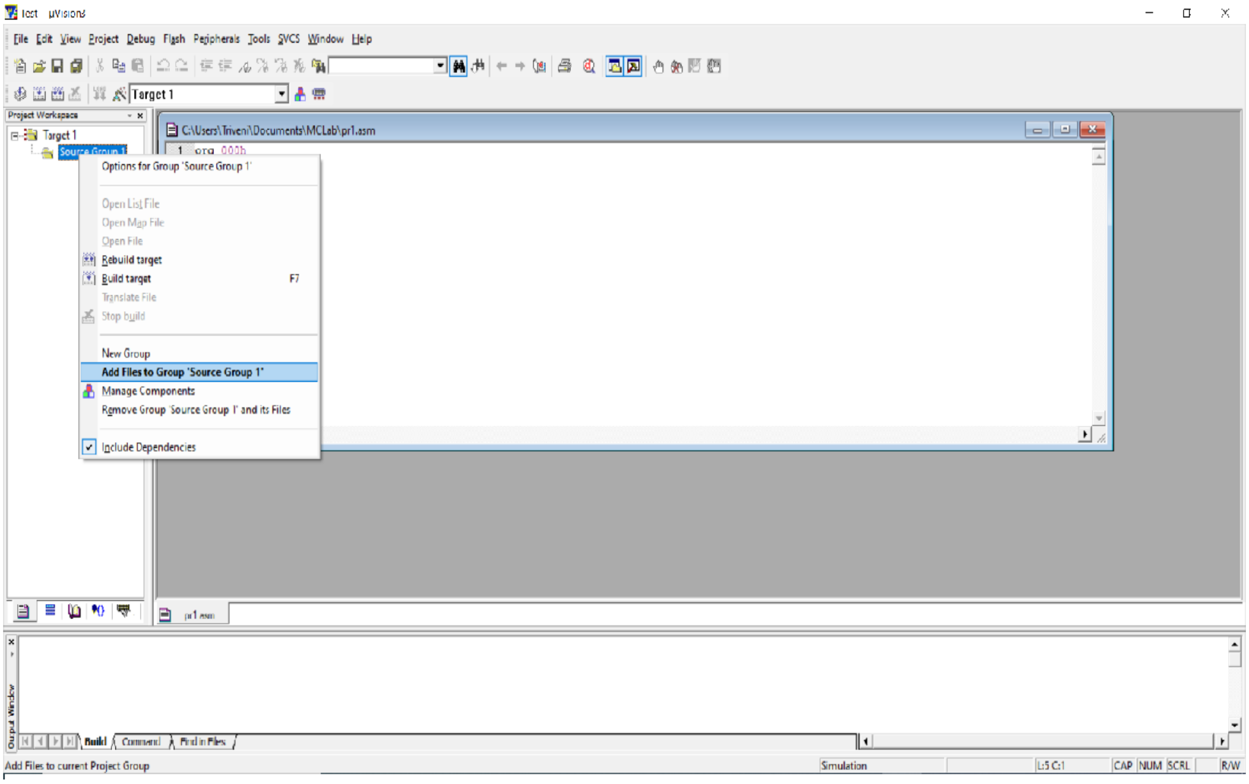
Save in: MCLab

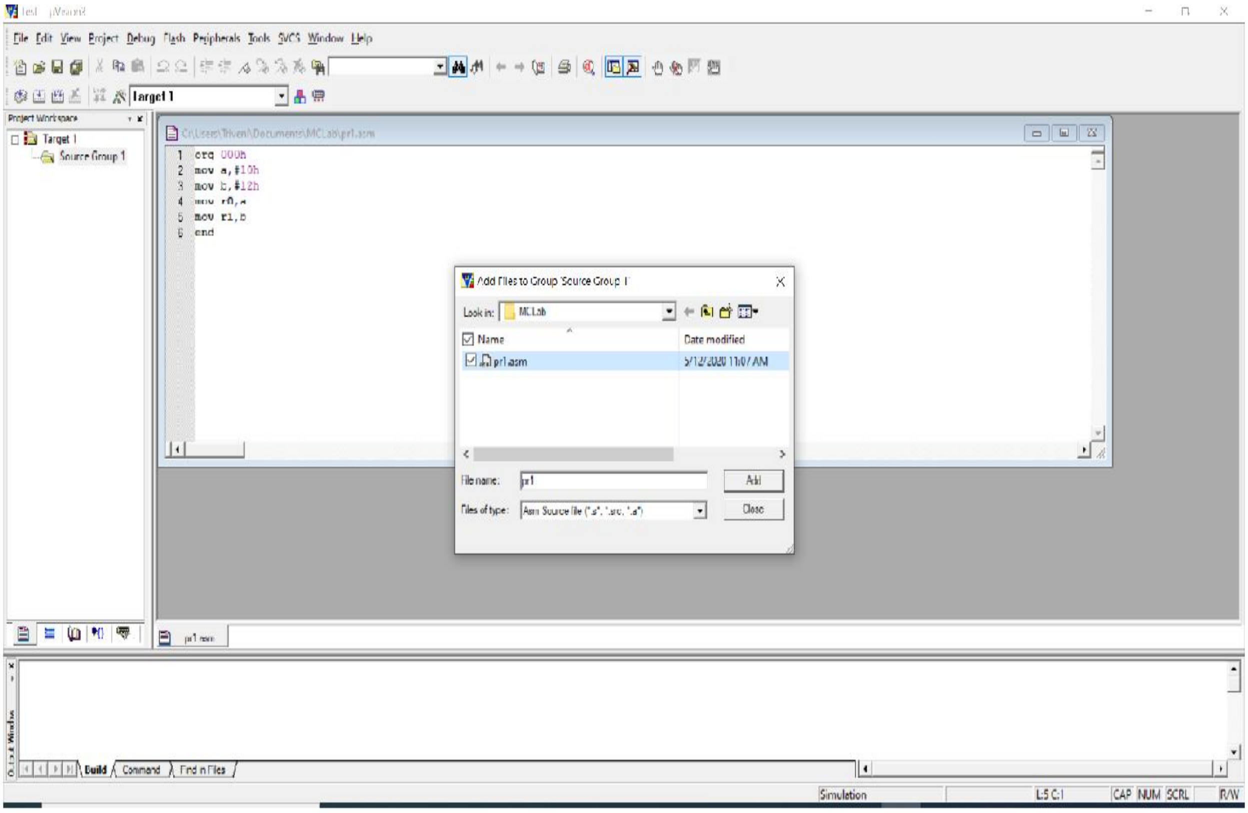
Name	Date modified
pr1.asm	5/12/2020 11:07 AM
pr1.LST	5/12/2020 11:08 AM
pr1	5/12/2020 11:08 AM
Test	5/12/2020 11:40 AM
Test.lnp	5/12/2020 11:40 AM
Test.M51	5/12/2020 11:40 AM

File name: P1.asm

Save as type: All Files (*.*)

STEP 7: Add your file to the target by right click on **source group1**.





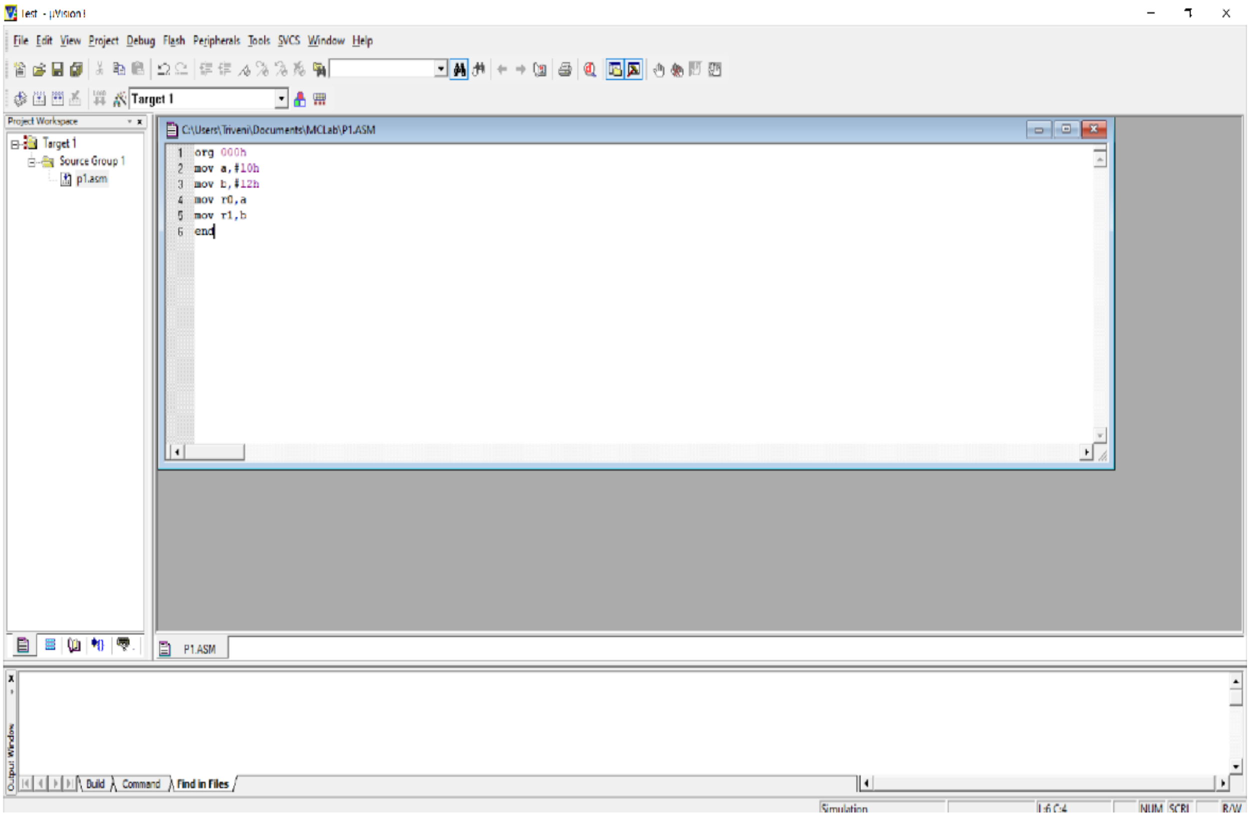
```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

Look in: MCLab

Name	Date modified
☑ pr1.asm	5/12/2020 11:07 AM

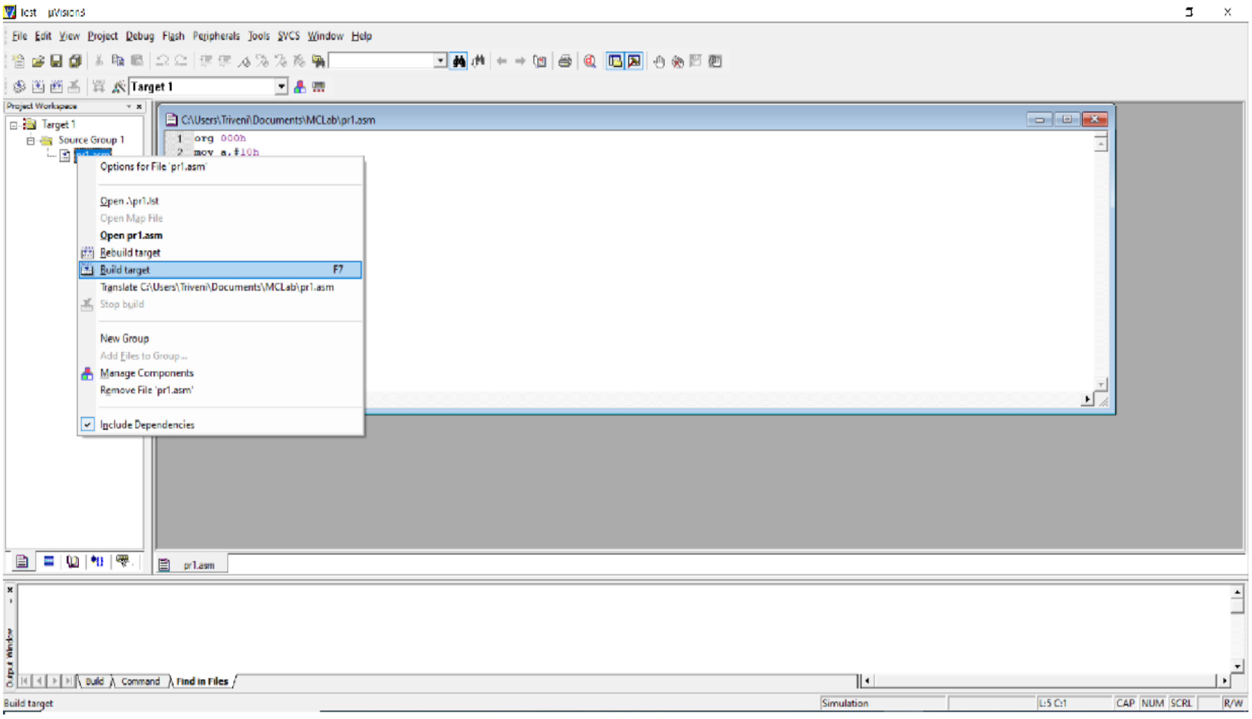
File name: pr1

Files of type: Asm Source file (*.s*; *.src; *.a*)

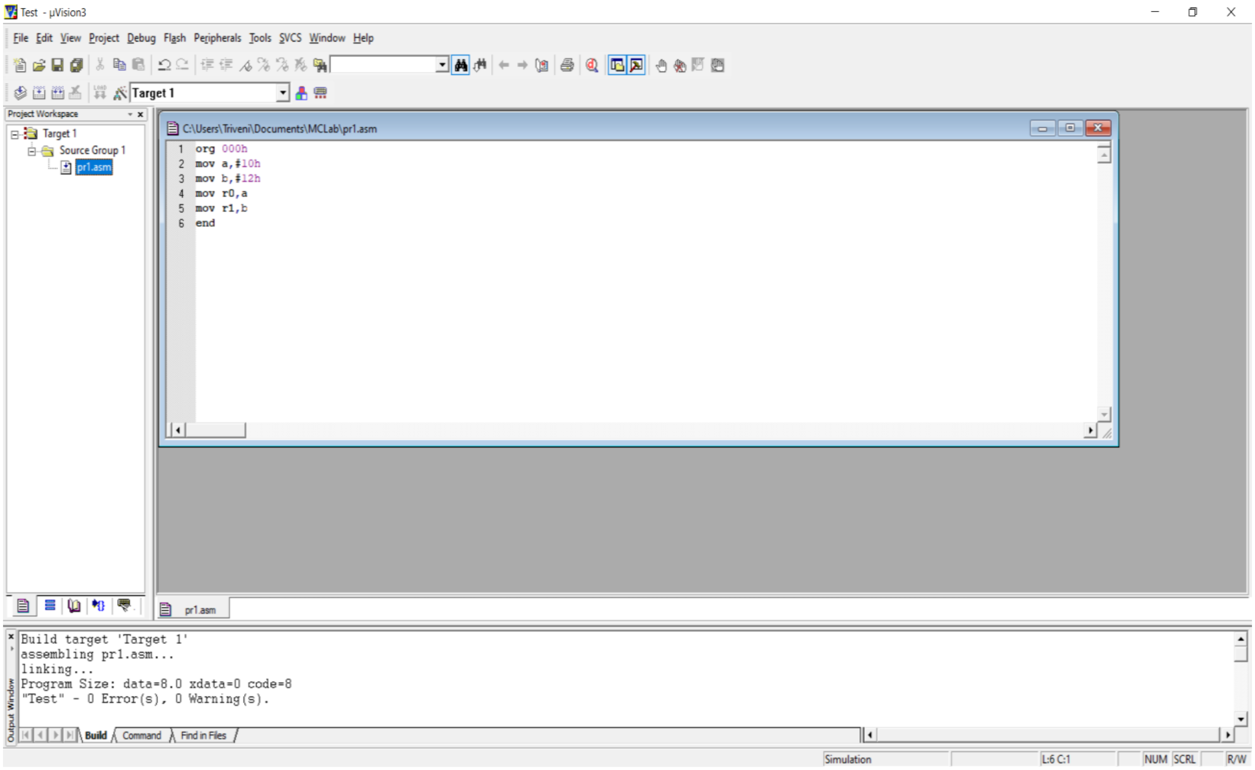


```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

STEP 8: Ones adding your file to the target check there are any syntax error in a program file by right clicking on file and selecting build/rebuild target.



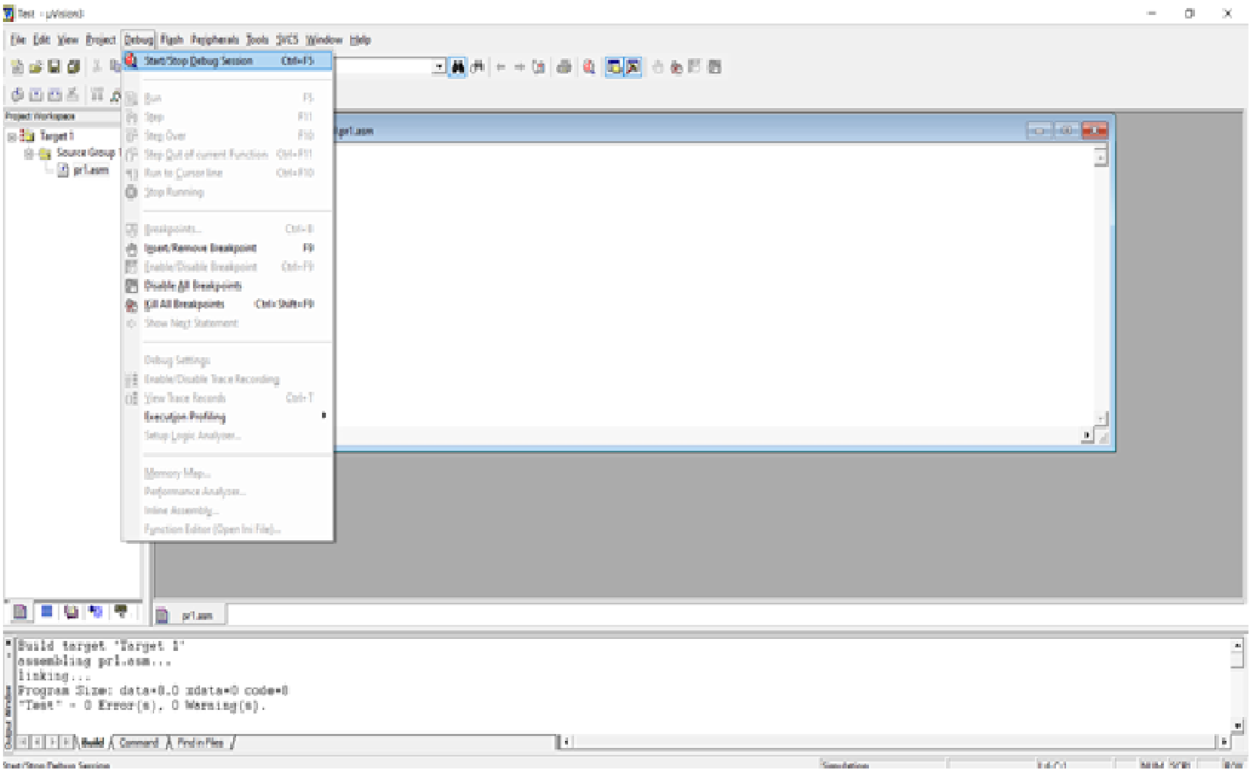
```
org 000h
mov a,#10h
```



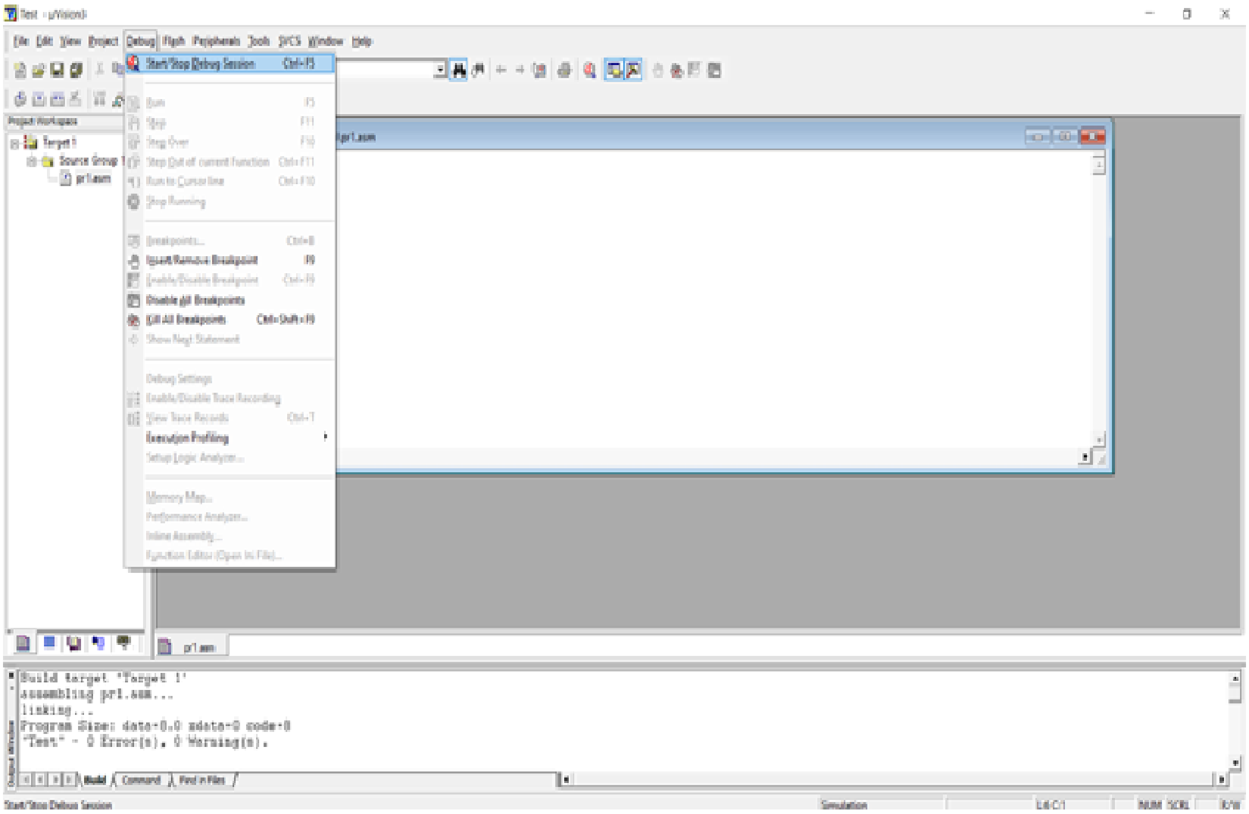
```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

```
Build target 'Target 1'
assembling pr1.asm...
linking...
Program Size: data=8.0 xdata=0 code=8
"Test" - 0 Error(s), 0 Warning(s).
```

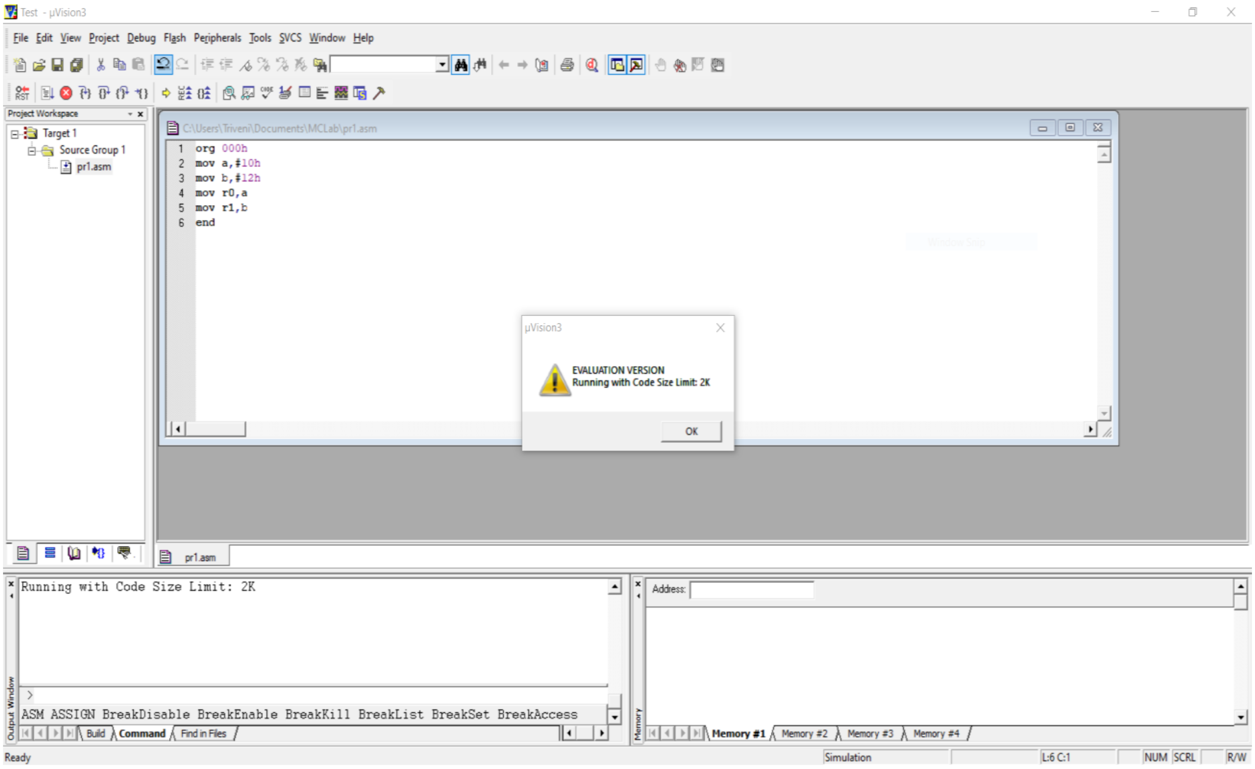
STEP 9: After getting **(0) Errors** then go to Debug menu and select start/stop debug session then say **OK**



```
Build target 'Target 1'
assembling pr1.asm...
linking...
Program Size: data=8.0 xdata=0 code=8
"Test" - 0 Error(s), 0 Warning(s).
```


Test - µVision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Start/Stop Debug Session Ctrl+F5
Run F5
Step F11
Step Over F10
Step Out of current function Ctrl+F11
Run to Cursor line Ctrl+F10
Stop Running
Breakpoints... Ctrl+B
Insert/Remove Breakpoint F9
Enable/Disable Breakpoint Ctrl+F9
Disable All Breakpoints
Kill All Breakpoints Ctrl+Shift+F9
Show Next Statement
Debug Settings...
Enable/Disable Trace Recording
View Trace Records Ctrl+T
Execution Profiling
Setup Logic Analyzer...
Memory Map...
Performance Analyzer...
Inline Assembly...
Function Editor (Open Ini File)...
Target 1
Source Group 1
pr1.asm
Build
Command
Find in Files
Start/Stop Debug Session
Simulation
L:6 C:1
NUM SCRL R/W

```
Build target 'Target 1'
assembling pr1.asm...
linking...
Program Size: data=8.0 xdata=0 code=8
"Test" - 0 Error(s), 0 Warning(s).
```

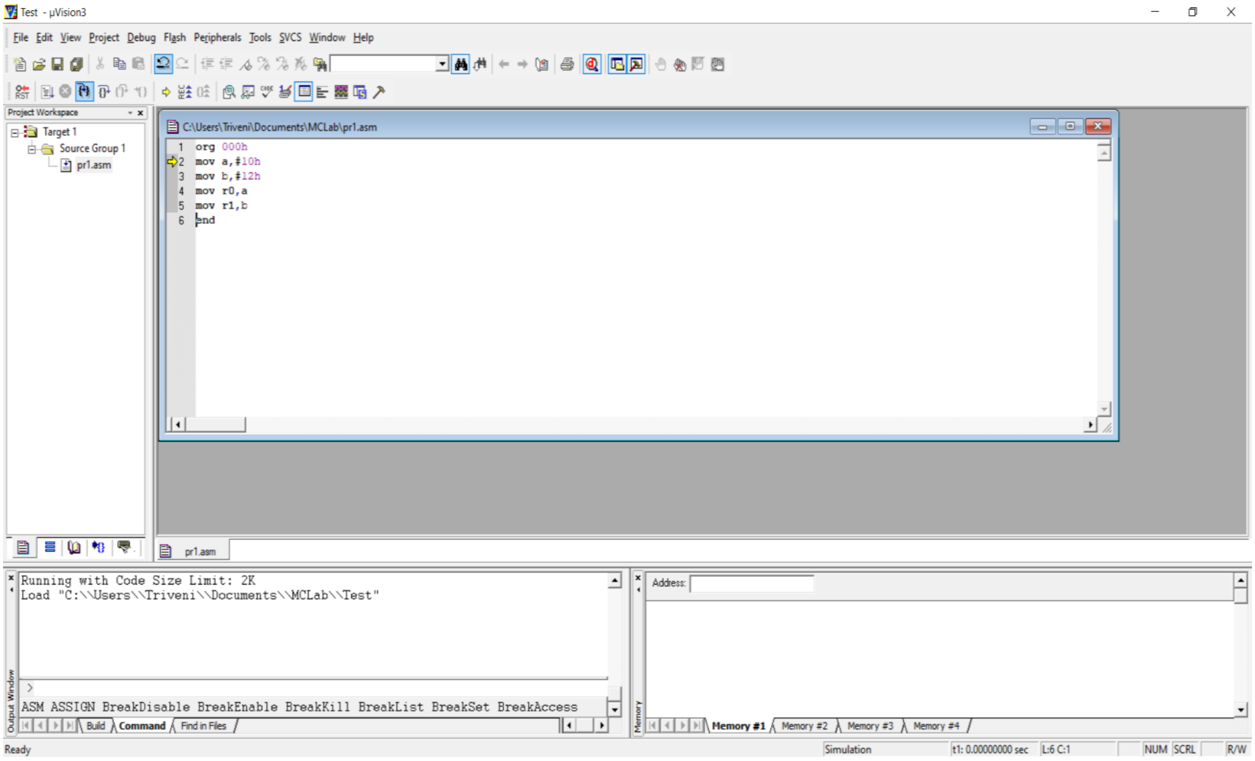


```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

EVALUATION VERSION
Running with Code Size Limit: 2K

```
Running with Code Size Limit: 2K
>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

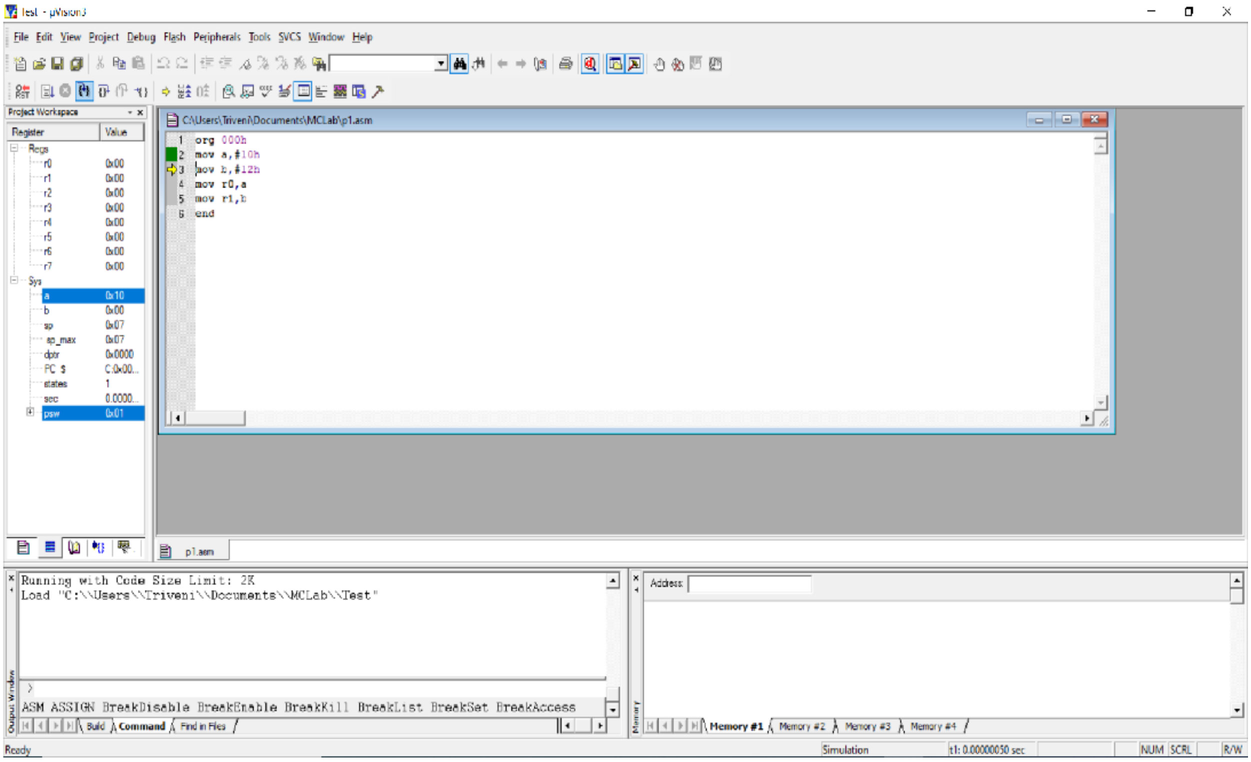
STEP 10: Now start executing your program in step into mode so that you can analyze the logic easily and get the correct output.



```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

```
Running with Code Size Limit: 2K
Load "C:\\Users\\Triveni\\Documents\\MCLab\\Test"
```

```
>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

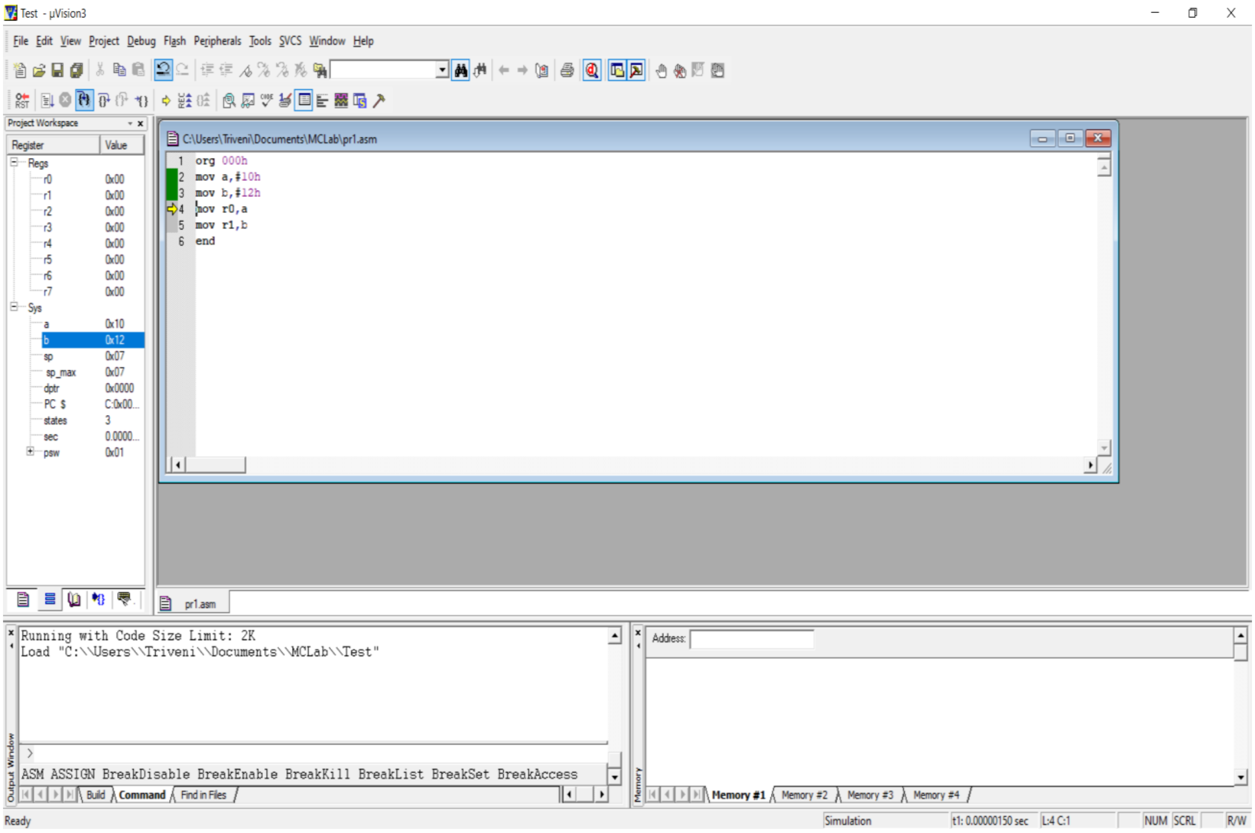


Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x10
b	0x00
sp	0x07
sp_max	0x07
dptr	0x0000
PC $	C:0x00...
states	1
sec	0.0000...
psw	0x01

```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

```
Running with Code Size Limit: 2K
Load "C:\\Users\\Triveni\\Documents\\MCLab\\Test"
```

```
>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

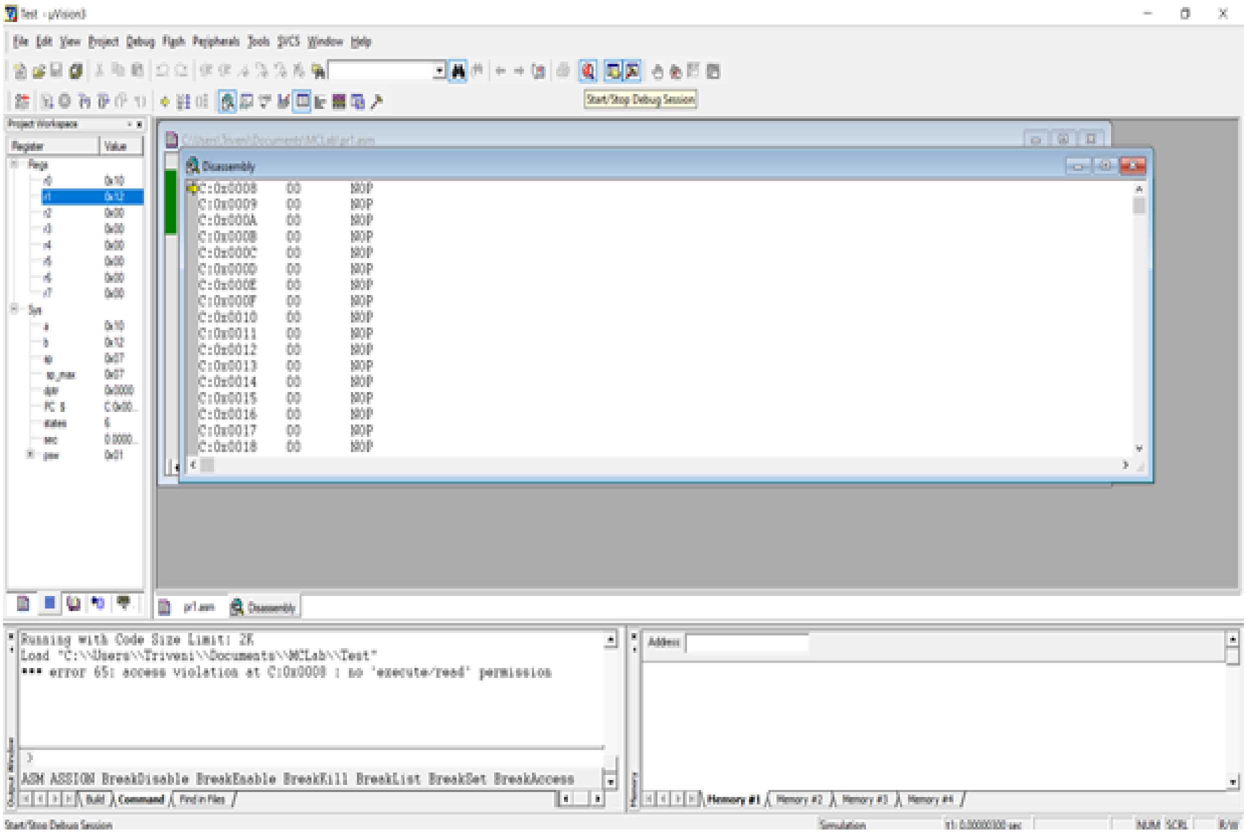


Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x10
b	0x12
sp	0x07
sp_max	0x07
dptr	0x0000
PC $	C:0x00...
states	3
sec	0.0000...
psw	0x01

```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

```
Running with Code Size Limit: 2K
Load "C:\\Users\\Triveni\\Documents\\MCLab\\Test"
>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

```
Ready          Simulation     t1: 0.00000150 sec   L:4 C:1     NUM SCRL   R/W
```

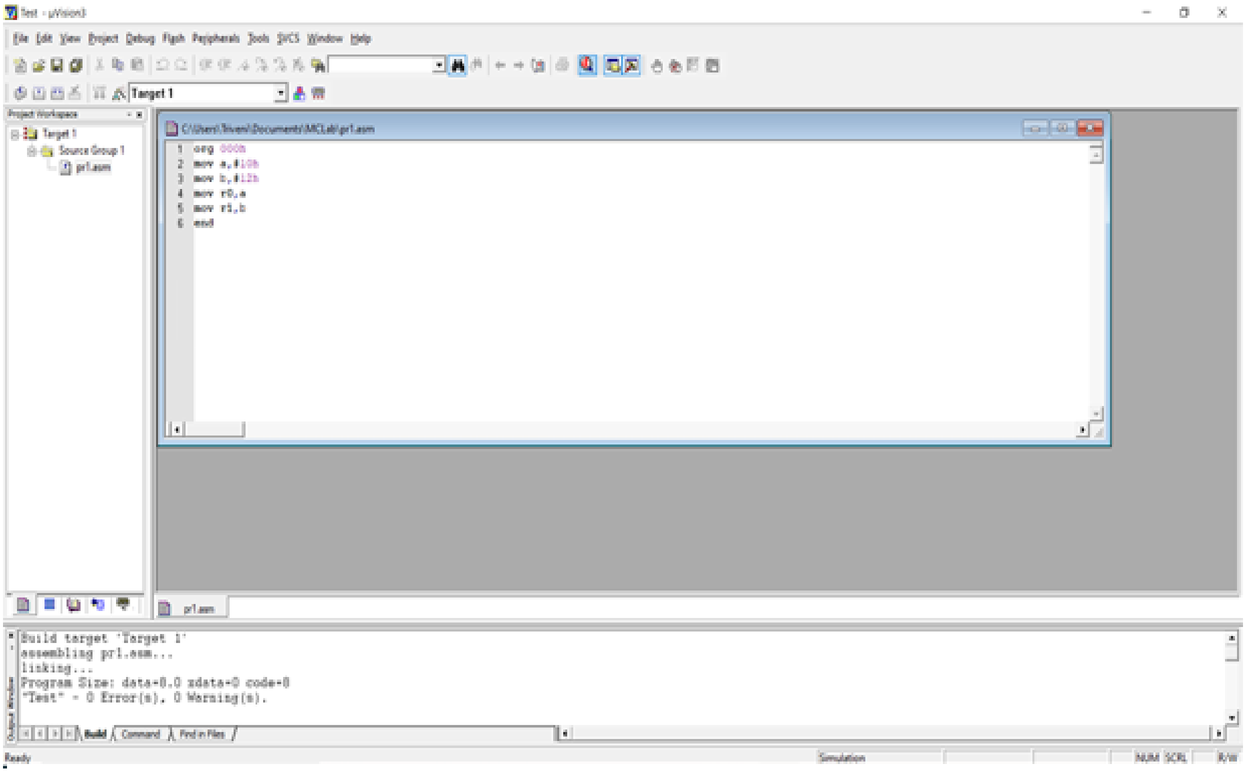


Register	Value
Regs	
r0	0x10
r1	0x12
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x10
b	0x12
sp	0x07
sp_max	0x07
dptr	0x0000
PC $	C:0x00...
states	6
sec	0.0000...
psw	0x01

```
C:0x0008    00    NOP
C:0x0009    00    NOP
C:0x000A    00    NOP
C:0x000B    00    NOP
C:0x000C    00    NOP
C:0x000D    00    NOP
C:0x000E    00    NOP
C:0x000F    00    NOP
C:0x0010    00    NOP
C:0x0011    00    NOP
C:0x0012    00    NOP
C:0x0013    00    NOP
C:0x0014    00    NOP
C:0x0015    00    NOP
C:0x0016    00    NOP
C:0x0017    00    NOP
C:0x0018    00    NOP
```

```
Running with Code Size Limit: 2K
Load "C:\\Users\\Triveni\\Documents\\MCLab\\Test"
*** error 65: access violation at C:0x0008 : no 'execute/read' permission
>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

```
Start/Stop Debug Session     Simulation     t1: 0.00000300 sec     NUM SCRL   R/W
```



```
org 000h
mov a,#10h
mov b,#12h
mov r0,a
mov r1,b
end
```

```
Build target 'Target 1'
assembling pr1.asm...
linking...
Program Size: data=8.0 xdata=0 code=0
"Test" - 0 Error(s), 0 Warning(s).
```

PROGRAM 1(a)

WALP to move a block of 10 bytes of data stored in the internal data memory from location 30H to the location starting from 40H of the internal data memory.

```
ORG 0000H
MOV R0, #30H
MOV R1, #40H
 MOV R2, #10
BACK: MOV A, @ R0
 MOV @R1, A
 INC RO
 INC R1
DJNZ R2, BACK
 SJMP $
 END
```

Before execution			
Source		Destination	
Memory Address	Data	Memory Address	Data
i:30H	11h	i:40H	xx
31H	12h	41H	xx
32H	13h	42H	xx
33H	14h	43H	xx
34H	15h	44H	xx
35H	16h	45H	xx
36H	17h	46H	xx
37H	18h	47H	xx
38H	19h	48H	xx
39H	1ah	49H	Xx
Register	Data		
R0			
R1			
R2			

After execution			
Source		Destination	
Memory Address	Data	Memory Address	Data
i:30H	11h	i:40H	11h
31H	12h	41H	12h
32H	13h	42H	13h
33H	14h	43H	14h
34H	15h	44H	15h
35H	16h	45H	16h
36H	17h	46H	17h
37H	18h	47H	18h
38H	19h	48H	19h
39H	1ah	49H	1ah
Register	Data		
R0			
R1			
R2			

PROGRAM 1(b)

WALP to move a block of 10 bytes of data stored in the internal data memory to the external data memory.

Starting address of the internal data memory: 30H.

Starting address of the external data memory: 100H.

```
       ORG 0000H
       MOV R0, #30H
        MOV DPTR, #100H
       MOV R2, #10
BACK:  MOV A, @ R0
       MOVX @DPTR, A
       INC RO
        INC DPTR
       DJNZ R2, BACK
       SJMP $
       END
```

Before execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H	11h	**X:100H**	xx
31H	12h	**101H**	xx
32H	13h	**102H**	xx
33H	14h	**103H**	xx
34H	15h	**104H**	xx
35H	16h	**105H**	xx
36H	17h	**106H**	xx
37H	18h	**107H**	xx
38H	19h	**108H**	xx
39H	1ah	**109H**	xx
Register	**Data**		
R0			
DPTR			
R1			

After execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H	11h	**X:100H**	11h
31H	12h	**101H**	12h
32H	13h	**102H**	13h
33H	14h	**103H**	14h
34H	15h	**104H**	15h
35H	16h	**105H**	16h
36H	17h	**106H**	17h
37H	18h	**107H**	18h
38H	19h	**108H**	19h
39H	1ah	**109H**	1ah
Register	**Data**		
R0			
DPTR			
R1			

PROGRAM 2 (a)

WALP to interchange the 10 bytes of data stored in the internal data memory from location 30H with 10 bytes of data stored from location 40H of the internal data memory

```
       ORG 0000H
       MOV R0, #30H
       MOV R1, #40H
       MOV R2, #10
BACK:  MOV A, @ R0
       XCH A,@R1
       MOV @R0, A
       INC RO
       INC R1
       DJNZ R2, BACK
       SJMP $
       END
```

Before execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H	11h	**i:40H**	aa
31H	12h	**41H**	bb
32H	13h	**42H**	cc
33H	14h	**43H**	dd
34H	15h	**44H**	ee
35H	16h	**45H**	ff
36H	17h	**46H**	99
37H	18h	**47H**	88
38H	19h	**48H**	77
39H	1ah	**49H**	66
Register	**Data**		
R0	30h		
R1	40h		
R2	0Ah		

After execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H	aa	**i:40H**	11h
31H	bb	**41H**	12h
32H	cc	**42H**	13h
33H	dd	**43H**	14h
34H	ee	**44H**	15h
35H	ff	**45H**	16h
36H	99	**46H**	17h
37H	88	**47H**	18h
38H	77	**48H**	19h
39H	66	**49H**	1ah
Register	**Data**		
R0	3Ah		
R1	4Ah		
R2	00h		

PROGRAM 2(b)

WALP to interchange the 10 bytes of data stored in the internal data memory with the 10 bytes of external data memory location.

Starting address of the internal data memory: 30H.

Starting address of the external data memory: 100H.

```
ORG 0000H
 MOV R0, #30H
MOV DPTR, #100H
 MOV R2, #10
BACK: MOVX A, @ DPTR
 XCH A, @R0
 MOVX @DPTR, A
 INC RO
 INC DPTR
DJNZ R2, BACK
 SJMP $
  END
```

Before execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H	11h	**X:100H**	aa
31H	12h	**101H**	bb
32H	13h	**102H**	cc
33H	14h	**103H**	dd
34H	15h	**104H**	ee
35H	16h	**105H**	ff
36H	17h	**106H**	99
37H	18h	**107H**	88
38H	19h	**108H**	77
39H	1ah	**109H**	66
Register	**Data**		
R0			
DPTR			
R1			

After execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H	aa	**X:100H**	11h
31H	bb	**101H**	12h
32H	cc	**102H**	13h
33H	dd	**103H**	14h
34H	ee	**104H**	15h
35H	ff	**105H**	16h
36H	99	**106H**	17h
37H	88	**107H**	18h
38H	77	**108H**	19h
39H	66	**109H**	1ah
Register	**Data**		
R0			
DPTR			
R1			

PROGRAM 3(a)

WALP to add 10 bytes of binary/hex numbers stored in the internal data memory from location 30H. Store the 16 bit sum at location 40 and 41H such that MS byte of sum is stored at 40H of the internal data memory.

```
ORG 0000H
MOV R0, #30H
 MOV R2, #10H
MOV R1, #00H
MOV A, R1
BACK:   ADD A, @ R0
JNC NEXT
INC R1
NEXT:   INC RO
DJNZ R2, BACK
MOV 40H, R1
MOV 41H, A
SJMP $
END
```

Before execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H		**i:40H**	XX
31H		**41H**	XX
32H			
33H			
34H			
35H			
36H			
37H			
38H			
39H			

After execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H		**i:40H**	XX
31H		**41H**	XX
32H			
33H			
34H			
35H			
36H			
37H			
38H			
39H			

PROGRAM 3(b)

WALP to add 10 bytes of BCD numbers stored in the internal data memory from location 30K. Store the 16 bit sum at locations 40H and 41H such that MS byte of sum stored at 40H of the internal data memory.

```
              ORG 0000H
               MOV R0, #30H
              MOV R2, #10H
              MOV R1, #00H
              MOV A, R1
     BACK:    ADD A,@R0
              DAA
              JNC NEXT
              INC R1
      NEXT:    INC R0
              BJNZ R2, BACK
              MOV 40H, R1
              MOV 41H, A
              SJMP $
              END
```

Before execution			
Source		Destination	
Memory Address	Data	Memory Address	Data
i:30H		i:40H	XX
31H		41H	XX
32H			
33H			
34H			
35H			
36H			
37H			
38H			
39H			

After execution			
Source		Destination	
Memory Address	Data	Memory Address	Data
i:30H		i:40H	XX
31H		41H	XX
32H			
33H			
34H			
35H			
36H			
37H			
38H			
39H			

PROGRAM 4(a)

WALP to add two multi byte numbers (at least 32 bits) stored in the internal data memory from location 30H and location 40H. Store the multi byte sum from location 50B of the internal data memory.

```
          ORG 0000H
          MOV R0, #30H
          MOV R1, #40H
          MOV R2, #50H
          MOV R3, #05H
          CLR C
   BACK:  MOV A,@R0
          ADDC A,@R1
          MOV 04H, R1
          MOV R1, 02H
          MOV @R1, A
          MOV 02H, R1
          MOV R1, 04H
          INC R0
          INC R1
          INCR2
   NEXT:  DJNZ R3, BACK
          MOV 01H, R2
          MOV A, #00H
          RLC A
          MOV @R1, A
          SJMP $
          END
```

Before execution					
Source 1		**Source 2**		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**	**Memory Address**	**Data**
i:30H		**i:40H**		**i = 50**	**XX**
31H		**41H**		**51H**	**XX**
32H		**42H**		**52H**	**XX**
33H		**43H**			
34H		**44H**			
35H		**45H**			

After execution					
Source 1		**Source 2**		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**	**Memory Address**	**Data**
i:30H		**i:40H**		**i = 50**	
31H		**41H**		**51H**	
32H		**42H**		**52H**	
33H		**43H**			
34H		**44H**			
35H		**45H**			

PROGRAM 4(b)

WALP to find the square and cube of an 8 bit binary number stored in the internal data memory location 30h

```
X EQU 30H
SQU EQU 31H
CUBE EQU 33H

ORG 0000H
MOV A,X
MOV R0,A
MOV B,A
MUL AB
MOV SQU,B
MOV SQU+1,A
MOV B,R0
MUL AB
MOV R1,A
MOV R2,B
MOV B,R0
MOV A,SQU
MUL AB
ADD A,R2
MOV R2,A
ADDC A,B
MOV R3,A
MOV CUBE,R3
MOV CUBE+1,R2
MOV CUBE+2,R1
SJMP $
END
```

Before execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H		**i:31H**	XX
		32H	XX
		33H	XX
		34H	XX
		35H	XX

After execution			
Source		**Destination**	
Memory Address	**Data**	**Memory Address**	**Data**
i:30H		**i:31H**	XX
		32H	XX
		33H	XX
		34H	XX
		35H	XX

PROGRAM 5(a)

WALP to find the largest element of an array of 10 bytes of the data stored in the internal data memory from location 30h.

```
ORG 0000H
MOV R2,#10
DEC R2
MOV R0,#30H
MOV A@R0
MOV B,R0
INC R0
BACK:    CLR C
 MOV R3,A
SUBB A@R0
JNC NEXT
MOV 03H,@R0
MOV B,R0
NEXT:MOV A,R3
INC R0
DJNZ R2,BACK
 MOV 41H,R3
 MOV 40H,B
SJMP $
 END
```

Before execution			
Source		Destination	
Memory Address	Data	Memory Address	Data
i:30H		i:40H	XX
31H		41H	XX
32H			
33H			
34H			
35H			
36H			
37H			
38H			
39H			

After execution			
Source		Destination	
Memory Address	Data	Memory Address	Data
i:30H		i:40H	XX
31H		41H	XX
32H			
33H			
34H			
35H			
36H			
37H			
38H			
39H			

PROGRAM 5(b)

WALP to sort the array of 10 eight bit number stored in the internal data memory location from location 30h

```
           ORG 0000H
    START:MOV R2,#10
           DEC R2
           MOV R0,#30H
           MOV R1,#31H
           MOV R3,#00H
    BACK:  CLR C
           MOV A,@R0
           SUBB A,@R1
           JC NEXT
           MOV A,@R0
           XCH A,@R1
           MOV @R0,A
           MOV R3,#01
           NEXT:INC R0
           INC R1
           DJNZ R2,BACK
           CJNE R3,#00,START
           SJMP $
           END
```

Before execution	
Source	
Memory Address	**Data**
i:30H	
31H	
32H	
33H	
34H	
35H	
36H	
37H	
38H	
39H	

After execution	
Destination	
Memory Address	**Data**
i:30H	
31H	
32H	
33H	
34H	
35H	
36H	
37H	
38H	
39H	

PROGRAM 6(a)

WALP to count the number of 1's in a byte which is be accepted from the port 0 and display the result in the port1.

```
        ORG 0000H
        MOV P0, #0FFH
BACK:   MOV A, P0
        MOV R0, #00H
        MOV R1, #08H
LOOP:   RLC A
        JNC NEXT
        INC R0
        NEXT: DJNZ R1, LOOP
        MOV P1, R0
        SJMP $
        END
```

Before Execution

Port-0	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
Port-1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

After Execution

Port-0								
Port-1							✓	✓

PROGRAM 6(b)

WALP to check whether the gives byte is a valid bit palindrome, accept the byte from the Port-0 and display **AA**h if valid else **00**H in the Port-1.

```
                ORG 0000H
                MOV P0,#0FFH
        BACK:   MOV A,P0
                MOV B,A
                MOV R0,#08
                ACALL REVERSE
                MOV A,30H
                CJNE A,B,NEXT
                MOV P1,#0AAH
                SJMP LAST
        NEXT:   MOV P1,#00H
        LAST:   SJMP BACK

REVERSE:        RLC A
                MOV R1,A
                MOV A,R2
                RRC A
                MOV R2,A
                MOV A,R1
                DJNZ R0,REVERSE
                MOV 30H,R2
                RET
                END
```

Before Execution

Port-0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
Port-1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

After Execution

Port-0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Port-1	✓		✓		✓		✓	

PROGRAM 7(a)

WALP to convert a two digits DECIMAL/8 bit BCD number to its equivalent 2 digit HEX /8 bit binary number.

```
ORG 0000H
MOV P0,#0FFH
MOV A,P0
MOV B,#10H
DIV AB
MOV R0,B
MOV B,#10
MUL AB
ADD A,R0
MOV P1,A
SJMP $
END
```

Before Execution

Port-0	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**
Port-1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

After Execution

Port-0	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**
Port-1					✓	✓	✓	✓

PROGRAM 7(b)

WALP to convert a two digits HEX/8 bit binary number to its binary number to its equivalent two digit DECIMAL/8bit BCD number.

```
ORG 0000H
MOV P0,#0FFH
MOV A,P0
MOV B,#10
DIV AB
MOV R0,B
MOV B,#10
DIV AB
MOV R2,A
MOV R1,B
MOV A,R1
SWAP A
ADD A,R0
MOV P1,R2
MOV P2,A
SJMP $
END
```

Before Execution

Port-0	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
Port-1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

After Execution

Port-0	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
Port-1				✓		✓		✓

PROGRAM 8(a)

WALP to convert an 8 bit BCD/2 digit DECIMAL number to its equivalent ASCII number.

```
ORG 0000H
MOV P0,#0FFH
 MOV A,P0
 MOV B,#10H
 DIV AB
 ORL A,#30H
 MOV P1,A
 ORL B,#30H
 MOV P2,B
 SJMP $
 END
```

Before Execution

Port-0	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
Port-1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Port-2	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

After Execution

Port-0	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**
Port-1			✓	✓		✓		✓
Port-2			✓	✓			✓	✓

PROGRAM 8(b)

WALP to convert the given two bytes of ASCII number to its equivalent 8 bit BCD/2 digit DCIMAL number.

```
ORG 0000H
MOV P0,#0FFH
 MOV A,P0
 MOV B,#10H
 DIV AB
 ORL A,#30H
 MOV P1,A
 ORL B,#30H
 MOV P2,B
 SJMP $
 END
```

Before Execution

Port-0	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**
Port-1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
Port-2	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

After Execution

Port-0	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**
Port-1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
Port-2		✓		✓			✓	✓

PROGRAM 9

The output the value AAH and 55H alternatively to the port 0 for every 2 sec.

```
          ORG 0000H
          MOV TMOD,#10H
          MOV A,#55H
      L1: MOV P0,A
          ACALL DELAY
          CPL A
          SJMP L1

          ORG 0050H
  DELAY:  MOV R0,#40
      L3: MOV TH1,#3CH
          MOV TL1,#0B0Hd
          SETB TR1
      L2: JNB TF1,L2
          CLR TR1
          CLR TF1
          DJNZ R0,L3
          RET
          END
```

PROGRAM 10

WALP to transfer the string of data using serial communication by configuring the serial port in mode_1 with a baud rate of 9600,1 step bit and 8 data bits.

```
            ORG 0000H
            MOV TMOD,#20H
            MOV TH1,#-3
            MOV SCON,#50H
            SETB TR1
            MOV DPTR,#DATA1
    START:  CLR A
            MOVC A,@A+DPTR
            JZ STOP
            MOV SBUF,A
       L1:  JNB T1,L1
            CLR T1
            INC DPTR
            SJMP START
     STOP:  NOP

            ORG 100H
   DATA1:DB "DEPARTMENT OF ECE",0;
            END
```

PR0GRAM 11

WALP to count the number of inputs to the counter_0 by configuring the timer as counter in mode_1 and display the count in ports.

```
        ORG 0000H
        MOV TMOD,#05H
        SETB P3.5
L1:     MOV TL0,#0
        MOV TH0,#0
        SETB TR0
L2:     MOV P2,TL0
        MOV P1,TH0
        JNB TF0,L2
        CLR TR0
        CLR TF0
        SJMP L1
        END
```

PR0GRAM 12

Let X,Y,Z refers to the contents of the memory location 30h,31h,and 32h respectively write an ALP to perform the following logical operations;
If X=00 perform the operation Y OR Z.
If X=01 perform the operation Y AND Z.
If X=02 perform the operation Y XOR Z.

```
                ORG 0000H
                MOV R1,30H
                MOV A,31H
                MOV B,32H
                CJNE R1,#00H,BRAND
                ORL A,B
                SJMP LAST
      BRAND:   CJNE R1,#01H,BRXOR
                ANL A,B
                SJMP LAST
      BRXOR:   CJNE R1,#02H,LAST
                XRL A,B
       LAST:   MOV 33H,A
                SJMP $
                END
```

PROGRAM 13

WALP to compare the bytes of data present at the memory location 21h and 22h and represent the result of comparison through the bits whose addresses are 00h and 01h.
If (21h)<(22h)then clear the bit at 01h and also set the bit at 00h.
If (21h)>(22h)then set the bit at 01h and also clear the bit at 00h.
If (21h)=(22h)then set the bit at 01h and also set the bit at 00h.

```
           ORG 0000H
           MOV A, 21H
           CLR C
           SUBB A, 22H
           JZ EQUAL
           JNC BIG
           SETB 00H
           CLR 01H
           SJMP LAST
      BIG: CLR 00H
           SETB 01H
           SJMP LAST
    EQUAL: SETB 00H
           SETB 01H
    LAST:   SJMP $
            END
```

Case 1:

Before execution	
Source	
Memory Address	**Data**
i:20H	*XX*
21H	*15*
22H	*14*

After execution	
Destination	
Memory Address	**Data**
i:20H	*01*
21H	*15*
22H	*14*

Case 2:

Before execution	
Source	
Memory Address	**Data**
i:20H	*XX*
21H	*14*
22H	*15*

After execution	
Destination	
Memory Address	**Data**
i:20H	*02*
21H	*14*
22H	*15*

Case3:

Before execution	
Source	
Memory Address	**Data**
i:20H	*XX*
21H	*15*
22H	*15*

After execution	
Destination	
Memory Address	**Data**
i:20H	*03*
21H	*15*
22H	*15*

PROGRAM 14

Let X,Y AND Z refers to the bits of the bit addressable memory whose addresses 00H,08H and 0FH respectively write an ALP to perform the following logical operation :
If X=0 perform If X=0 perform the operation Y OR Z.
If X=0 perform If X=0 perform the operation Y AND Z.

```
          ORG 0000H
           MOV C, 08H
           JB 00H, NEXT
           ORL C, 0FH
           SJMP LAST
NEXT:  ANL C, 0FH
LAST:  MOV 10H, C
           SJMP $
            END
```

PROGRAM 15

Write an ALP to stimulate the Boolean expression Y=A+BC

```
        ORG 0000H
        MOV P1, #07H
NEXT:   MOV C,P1.0
        ANL C, /P1.1
        ORL C, P1.2
        MOV P1.7, C
        SJMP NEXT
        END
```

PROGRAM 16(a)

To implement an 8 bit binary/2 digits hex up counter. The counter has to count from 00H to FFH.

```
            ORG 0000H
            CLR A
BACK:       MOV P1, A
            ACALL DELAY
            ACALL DELAY
            INC A
            JNZ BACK
            SJMP $
```

Delay routine:

```
                ORG 50H
      DELAY:    MOV R1, #030H
         L3:    MOV R2, #0FFH
         L2:    MOV R3, #0FFH
         L1:    DJNZ R3, L1
                DJNZ R2, L2
                DJNZ R1, L3
                RET
                END
```

PROGRAM 16(b)

WALP to implement an 8 bit binary/2 digits hex down counter.the counter has to count from FFH to 00H.

```
               ORG 0000H
               CLR A
               CPL A
      BACK:   MOV P1, A
               ACALL DELAY
               ACALL DELAY
               DECA
              JNZ BACK
               MOV P1, A
               SJMP $

Delay routine:
               ORG 50H
      DELAY: MOV R1,#030H
         L3:   MOV R2,#0FFH
         L2:   MOV R3,#0FFH
         L1:   DJNZ R3,L1
               DJNZ R2,L2
               DJNZ R1,L3
               RET
               END
```

PROGRAM 17(a)

WALP to implement a 2 digits decimal/8 bit BCD up counter. The counter has to count from 00-99.

```
             ORG 0000H
             CLR A
      BACK:  MOV P1, A
             ACALL DELAY
             ACALL DELAY
             ADD A,#01
             DA A
             JNZ BACK
             SJMP $
```

Delay routine:

```
             ORG 50H
     DELAY:  MOV R1, #030H
       L3:   MOV R2, #0FFH
       L2:   MOV R3, #0FFH
       L1:   DJNZ R3, L1
             DJNZ R2, L2
             DJNZ R1, L3
             RET
             END
```

PROGRAM 17(b)

WALP to implement a 2 digits decimal/8 bit BCD down counter. The counter has to count from 99-00.

```
                ORG 0000H
                MOV A, #99H
        BACK:   MOV P1, A
                ACALL DELAY
                ACALL DELAY
                ADD A, #99H
                DA A
                JNZ BACK
                MOV P1, A
                SJMP $
```

Delay routine:

```
                ORG 50H
        DELAY:  MOV R1, #030H
           L3:  MOV R2, #0FFH
           L2:  MOV R3, #0FFH
           L1:  DJNZ R3, L1
                DJNZ R2, L2
                DJNZ R1, L3
                RET
                END
```

PART-B

EMBEDEDDED C-PROGRAMMING

PROGRAM 1:

//PROGRAM TO ACCEPT THE INPUT FROM PORT 0 AND SENT TO THE PORT 2.

```
#include<reg52.h>
Void main (void)
{
P0=0xFF;
While (1)
{
P2=P0;
}
}
```

PROGRAM 2:

//PROGRAM TO DISPLAY BCD UP COUNTING.

//Connections K 11 with the K20 and connect K 12 with K21

```
#include<reg52.h>
#define DEL 3000
#define LOWER 0x00
#define UPPER 100
unsigned char binbcd(unsgined char);
void delay(unsgined int del)
{
while(del--);
}
void main(void)
{
unsgined char val;
while(1)
{
for (val=LOWER;val<UPPER;v++)
{
P0=binbcd(val);
delay(DEL);
}
}
}
unsigned char bincd(unsgined char i)
{
return(((i/10)<<4)|(i%10));
}
```

PROGRAM 3(a):

//PROGRAM TO GENERATE TRIANGULAR WAVEFORM USING DAC.

```
#include<reg52.h>
 void main(void)
 {
 signedint ramp;
 while(1)
 {
  for(ramp=0*00;ramp<=0*FF;ramp++)
 {
  P0=ramp;
 }
 for(ramp=0*FF;ramp>=0*00;ramp--)
 {
 P0=ramp;
 }
 }
 }
```

PROGRAM 3(b):

//PROGRAM TO GENERATE SQUARE WAVEFORM USING DAC

```
.#include<reg52.h>
#define DELAY 512
void delay(unsigned int del)
{
while(del--)
}
void main(void)
{
while(1)
{
P0=0*00;
delay(DELAY);
P0=0*FF;
delay(DELAY);
}
}
```

PROGRAM 4:

//PROGRAM TO DISPLAY THE KEY PRESSED.

//Connections: 1)Connect K 22 with the K 29 and connect K 23 with K 28.
2)Connect K 20 with the K 13 and connect K 21 with K 14.

```
#include<reg52.h>
#define DEL 3000
void delay1(unsigned int value);
void main()
{
unsigned char row, columnandrow,key;
unsigned char excite[4]={0*fe,0*fd,0*fb,0*f7);
unsigned char
value[16]={0*ee,0*de,0*be,0*7e,0*ed,0*dd,0*bd,0*7d,0*eb,0*db,0*bb,0*7b,0*e7,0*d7,0*b7,0*
77);
while(1)
{
for(row=0x00;row<=0x03;++row)
{
P2=exite[row];
columnandrow=P2;
for(key=00;key<16;key++)
{
if(columnnandrow==value[key])
break;
}
if(key<16)
{
P0=key;
}
delay1(DELAY);
}
}
}
void delay1(unsigned int value)
{
while(value--)
}
```

PROGRAM 5(a):

//PROGRAM TO CONTROL THE STEPPER MOTOR.

//Connections: 1)Connect K 20 with the K 35. 2)Connect K 37 with the Stepper motor.

```
#include<reg52.h>
#define DELAY 1000
void delay1(unsigned int value)
{
while(value--);
}
void main(void)
{
unsigned char val[]={0x0A.0x06,0x05,0x09}, i;
while(1)
{
for(i=0; i<=0x03; i++)
{
P0=val[i];
delay1(DELAY);
}
}
}
```

PROGRAM 5(b):

//PROGRAM TO CONTROL THE STEPPER MOTOR FOR THE REQUIRED NO OF ROTATIONS.

//Connections: 1)Connect K 20 with the K 35. 2)Connect K 37 with the Stepper motor.

```
#include<reg52.h>
#define DELAY 1000
void delay1(unsigned int value)
{
while(value--);
}
void main(void)
{
unsigned char val[]={0x0A.0x06,0x05,0x09},i,j;
{
for(i=0; j<=48; j++)
{
P0=val[i];
delay1(DELAY);
}
```

```
 }
 while(1);
}
```

PROGRAM 6:

//PROGRAM ON LCD DISPLAY

```
#include<reg52.h>
Sbit rs =P2^0;
Sbit rw =P2^1;
Sbit en =P2^2;
Void lcd(unsigned char,bit);
Void delay1(unsigned int del)
{
While(--del);
}
Void lcd_init()
{
Lcd(0x38,0);
Lcd(0x38,0);
Lcd(0x38,0);
Lcd(0x38,0);

Lcd(0x01,0);
Lcd(0x0E,0);
Lcd(0x06,0);
}
Void main(void)
{
Unsigned char arr[15]={‘G’,’O’,’O’,’D’,’B’,’A’,’D’,’U’,’G’,’L’,’Y’,}

Lcd_init();

While(1)
{
Lcd(0x80,0);
For(i=0;i<=13;i++)
Lcd(arr[i],1);
}
}
```

APPENDIX A

GENERAL QUESTIONS

1. Upon reset, all ports of the 8051 are configured as______________(output, input).
2. Which ports of the 8051 have internal pull-up resistors?
3. Which ports of the 8051 require the connection of external pull-up resistors in order to be used for I/O? Show the drawing for the connection.
4. In the 8051, explain why we must write "1" to a port in order for it to be used for input.
5. Explain why we need to buffer the switches used as input in order to avoid damaging the 8051 port.
6. How does the LCD distinguish data from instruction codes when receiving information at its data pin?
7. To send the instruction code 01 to clear the display, we must make RS =____.
8. To send letter 'A' to be displayed on the LCD, we must make RS =_____.
9. What is the purpose of the E line? Is it an input or an output as far as the LCD is concerned?
10. When is the information (code or data) on the LCD pin latched into the LCD?
11. Indicate the direction of pins WR, RD, and INTR from the point of view of the 8051.
12. Give the three steps for converting data and getting the data out of the ADC804. State the status of the CS, RD, INTR, and WR pins in each step.
13. Assume that $V_{ref}/2$ is connected to 1.28 V. Find the following.
    13.1. step size
    13.2. maximum range for V_{in}
    13.3. D7 - D0 values if V_{in} = 1.2 V
    13.4. V_{in} if D7 - D0 = 11111111
    13.5. V_{in} if D7 - D0 = 10011100
14. Assume that $V_{ref}/2$ is connected to 1.9 V. Find the following.
    14.1. step size
    14.2. maximum range for V_{in}
    14.3. D7 - D0 values if V_{in} = 2.7 V
    14.4. V_{in} if D7 - D0 = 11111111
    14.5. V_{in} if D7 - D0 = 11011101
15. The ADC804 is a(n)_____-bit converter.
16. To get step size of 2 mV, what is the value for Vref/2?
17. What is a transducer?
18. What is the form of the transducer output?
19. What is preprocessing of transducer signals to be fed into an ADC called?
20. The LM35 and LM34 produce a________mV output for every degree of change in temperature.
21. The LM35/LM34 is a_____________(linear, nonlinear) device. Discuss the advantages of linear devices and of nonlinear devices.
22. Explain signal conditioning and its role in data acquisition.
23. What is the maximum frequency that can be generated using Mode 1 if the crystal frequency is 11.0592 MHz? Show your calculation.

24. What is the maximum frequency that can be generated using Mode 2 if the crystal frequency is 11.0592 MHz? Show your calculation.
25. What is the lowest frequency that can be generated using Mode 1 if the crystal frequency is 11.0592 MHz? Show your calculation.
26. What is the lowest frequency that can be generated using Mode 1 if the crystal frequency is 11.0592 MHz? Show your calculation.
27. In mode 1, when is TFx set to high?
28. In mode 2, when is TFx set to high?
29. The 8051 TxD and RxD signals_________(are, are not) TTL-compatible.
30. In this lab, what is the role of the MAX233 (MAX232) chip?
31. With XTAL=11.0592 MHz, what is the maximum baud rate for the 8051?
32. Show how to achieve the maximum baud rate
33. What is the role of TI and RI?
34. True or false. The 8051 can transfer data in full-duplex.
35. For full duplex, what are the absolute minimum signals needed between the 8051 and the PC? Give their names.
36. What is a step angle? Define steps per revolution.
37. If a given stepper motor has a step angle of 5 degrees, find the number of steps per revolution.
38. Give the four sequences for counter clockwise if it starts with 10011001 (binary).
39. Using the "RL A" instruction, show the four-step sequences if the initial step is 0011 (binary).
40. Give the number of times the four-step sequence must be applied to a stepper motor to make a 100-degree move if the motor has a 5-degree step angle. Also fill in the characteristics for your motor below.

    40.1. Step angle________Degree of movement per 4-step sequence ________

    40.2. Steps per revolution ______ Number of rotor teeth ______________

    40.3. What is the purpose of generating the truth table for a given keyboard?
41. What is the purpose of grounding each row in keyboard interfacing?
42. What is the input to the microcontroller from column if no key is pressed?
43. True or false. In our N x M matrix keypad program we cannot press two keys at the same time.
44. In your program in, how is the key press detected?
45. In your program in, how is a key press identified?
46. Explain the role of the C/T bit in the TMOD register.
47. How is the 8051 used as an event counter to count an external event?
48. If timer/counter 0 is used as an event counter, what is the maximum count for the following modes.

    48.1. Mode 1 48.2 Mode 2
49. Indicate which pin is used for the following.

    49.1. timer/counter 0 49.2 timer/counter 1
50. If timer/counter 0 is used in mode 1 to count an external event, explain when TF0 is set to high.
51. If timer/counter 1 is used in mode 2 to count an external event, explain when TF0 is set to high.
52. Indicate the direction of pins ALE, SC, EOC, and OE from the point of view of the ADC808/809.
53. Give the steps for converting data and getting the data out of the ADC809. State the status of the SC and EOC pins in each step.
54. Give the role of signals ALE, A, B, and C in selecting the ADC channel.
55. In the ADC809 assume that V_{ref} is connected to 2.56 V. Find the following.

    55.1. step size

    55.2. maximum range for V_{in}

    55.3. D7 - D0 values if V_{in} = 1.2 V

    55.4. V_{in} if D7 - D0 = 11111111

    55.5. V_{in} if D7 - D0 = 10011100
56. In the ADC809 assume that V_{ref} is connected to 5V. Find the following.

    56.1. step size

    56.2. maximum range for V_{in}

    56.3. D7 - D0 values if Vin = 2.7 V

    56.4. V_{in} if D7 - D0 = 11111111

    56.5. V_{in} if D7 – D0 = 11011101

57. In connecting ADC808/809 to an 8051, indicate the direction of pins ALE, SC, EOC, and OE from the point of view of the 8051.
58. Define the following terminology in DAC.

    58.1. resolution
    58.2. full-scale voltage output
    58.3. settling time

59. For your circuit, find V_{out} for the following inputs.

    59.1. 11001100
    59.2. 10001111

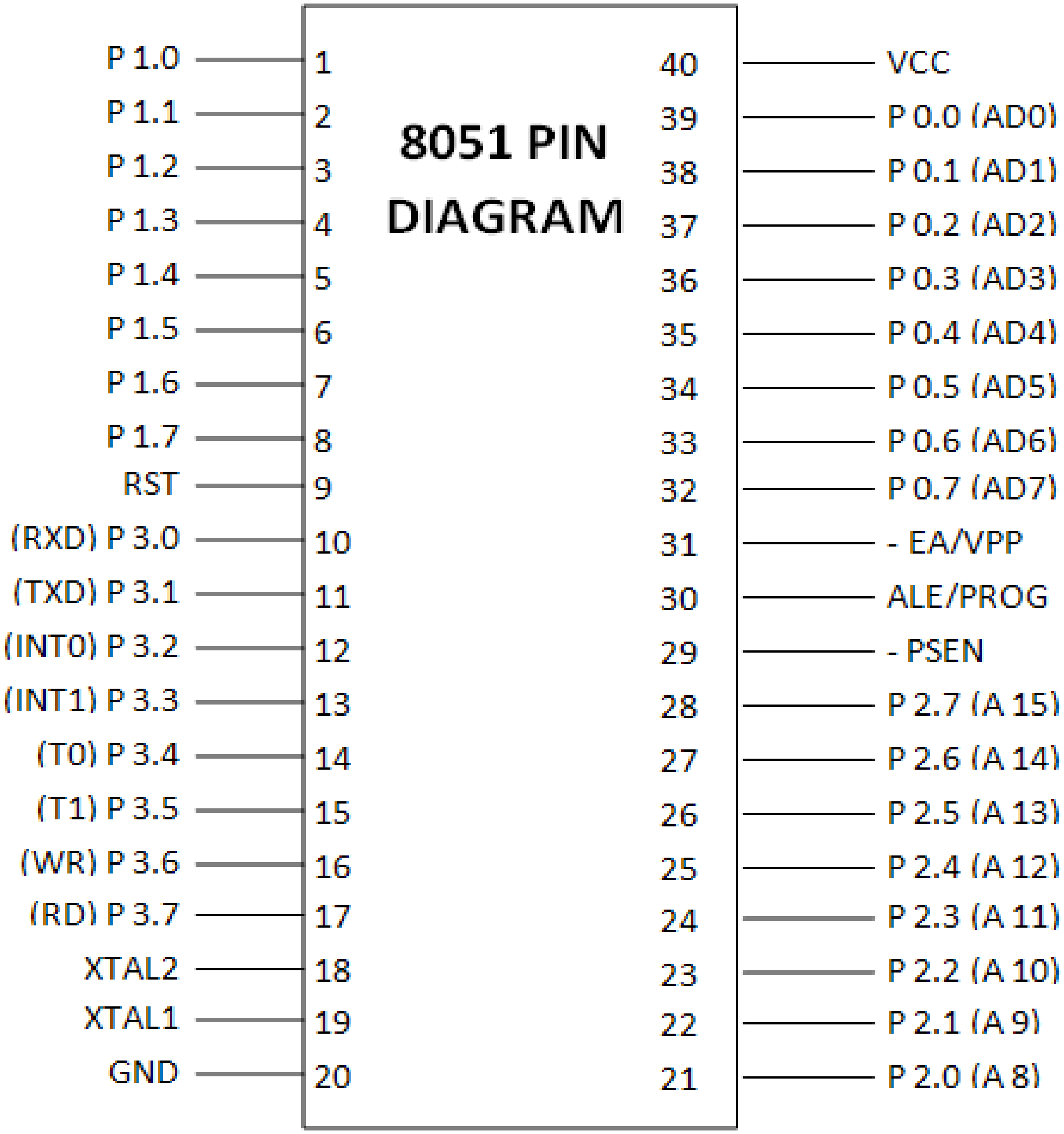
60. To get a smaller step size, we need DAC with_________(more, less) data bit inputs.
61. In Figure 13-7 of the textbook, assume that R = 2.5 K ohms. Calculate V_{out} for the following binary inputs.

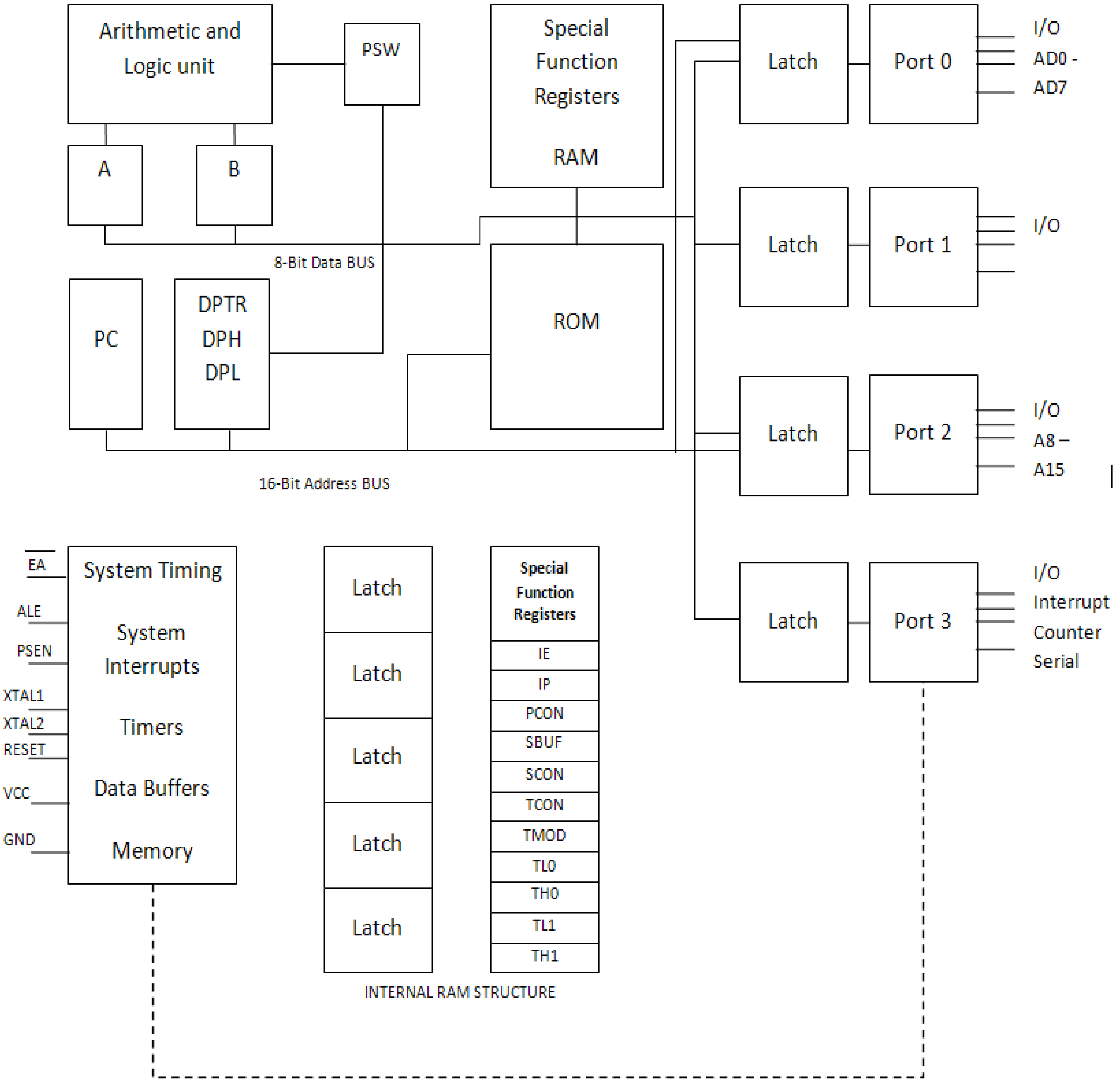
    61.1. 11000010
    61.2. 01000001
    61.3. 00101100
    61.4. 11111111

62. Name all of the interrupts in the 8051 and their vector table addresses.
63. In timer mode 1, indicate when TF0 causes the interrupt.
64. In timer mode 2, indicate when TF0 causes the interrupt.
65. On reset, INT0 (and INT1) are__________(edge, level) triggered.
66. On reset, which interrupt has the highest priority?
67. True or False. There is only a single interrupt for the serial data transfer.

APPENDIX B

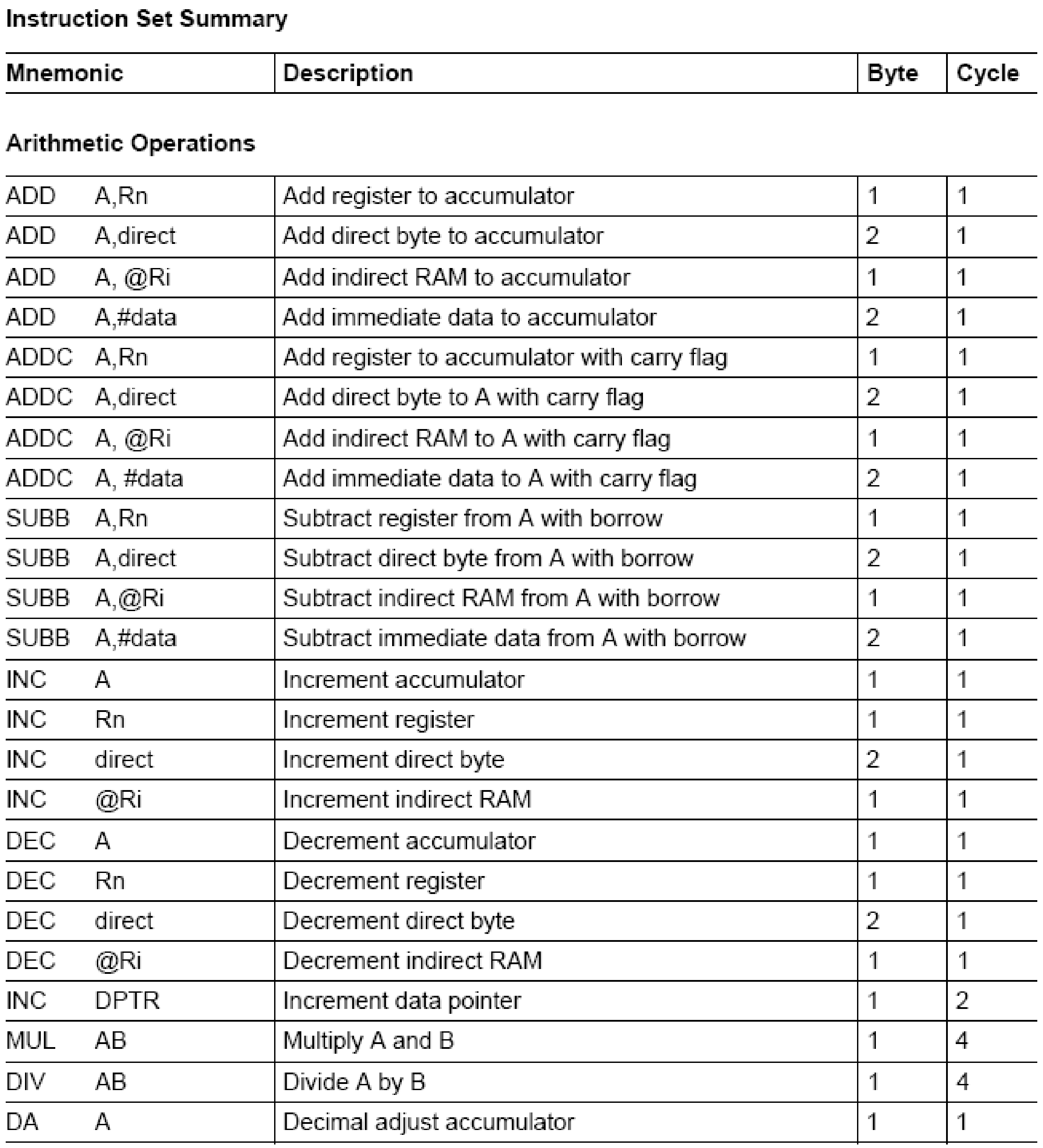
8051 PIN DIAGRAM AND ARCHITECTURE




Arithmetic and Logic unit
PSW
Special Function Registers
RAM
Latch
Port 0
I/O
AD0 -
AD7
A
B
8-Bit Data BUS
Latch
Port 1
I/O
PC
DPTR
DPH
DPL
ROM
Latch
Port 2
I/O
A8 –
A15
16-Bit Address BUS
EA
ALE
PSEN
XTAL1
XTAL2
RESET
VCC
GND
System Timing
System Interrupts
Timers
Data Buffers
Memory
Latch
Latch
Latch
Latch
Latch
Special Function Registers
IE
IP
PCON
SBUF
SCON
TCON
TMOD
TL0
TH0
TL1
TH1
INTERNAL RAM STRUCTURE
Latch
Port 3
I/O
Interrupt
Counter
Serial

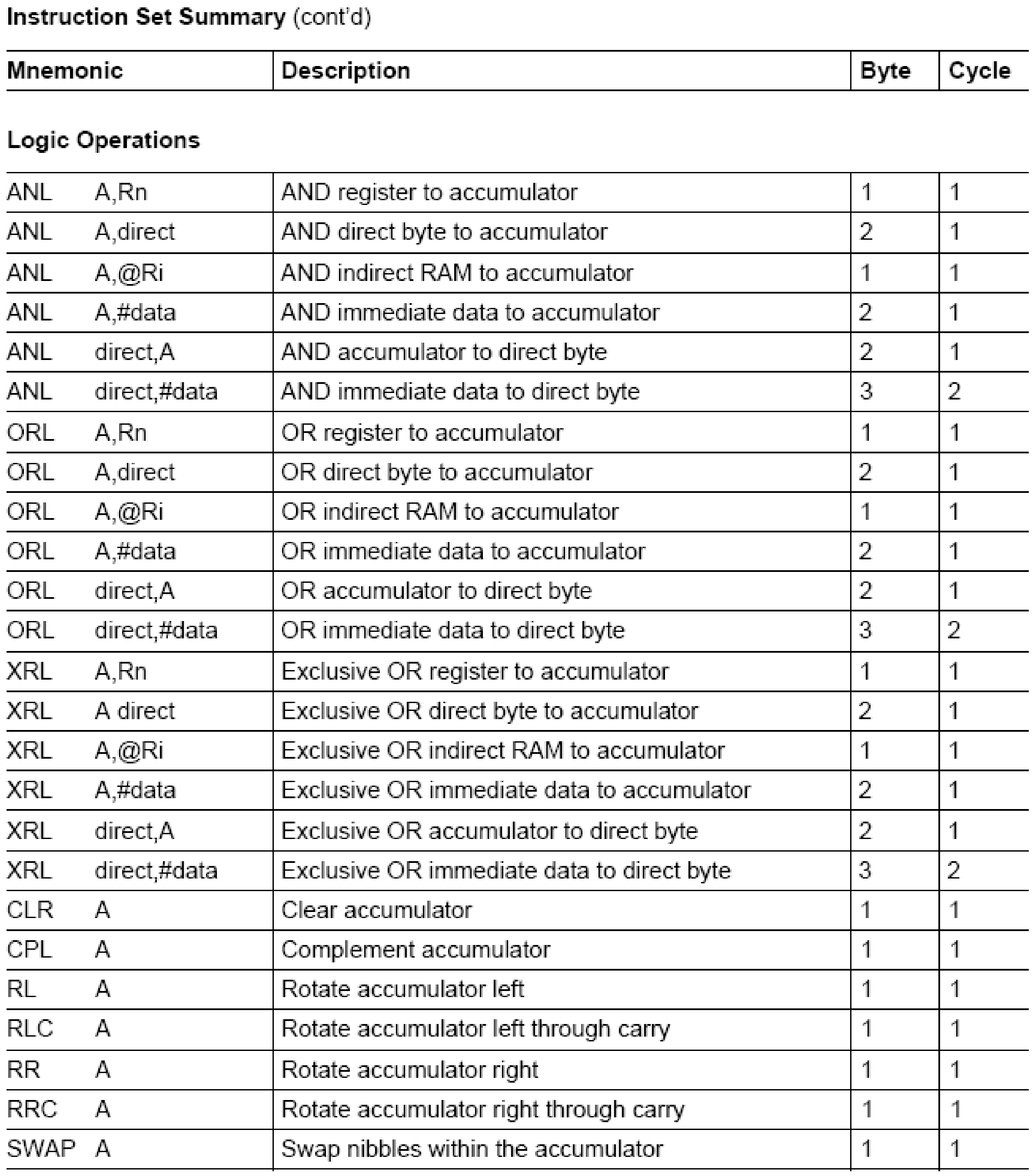
APPENDIX C

INSTRUCTION SET SUMMARY



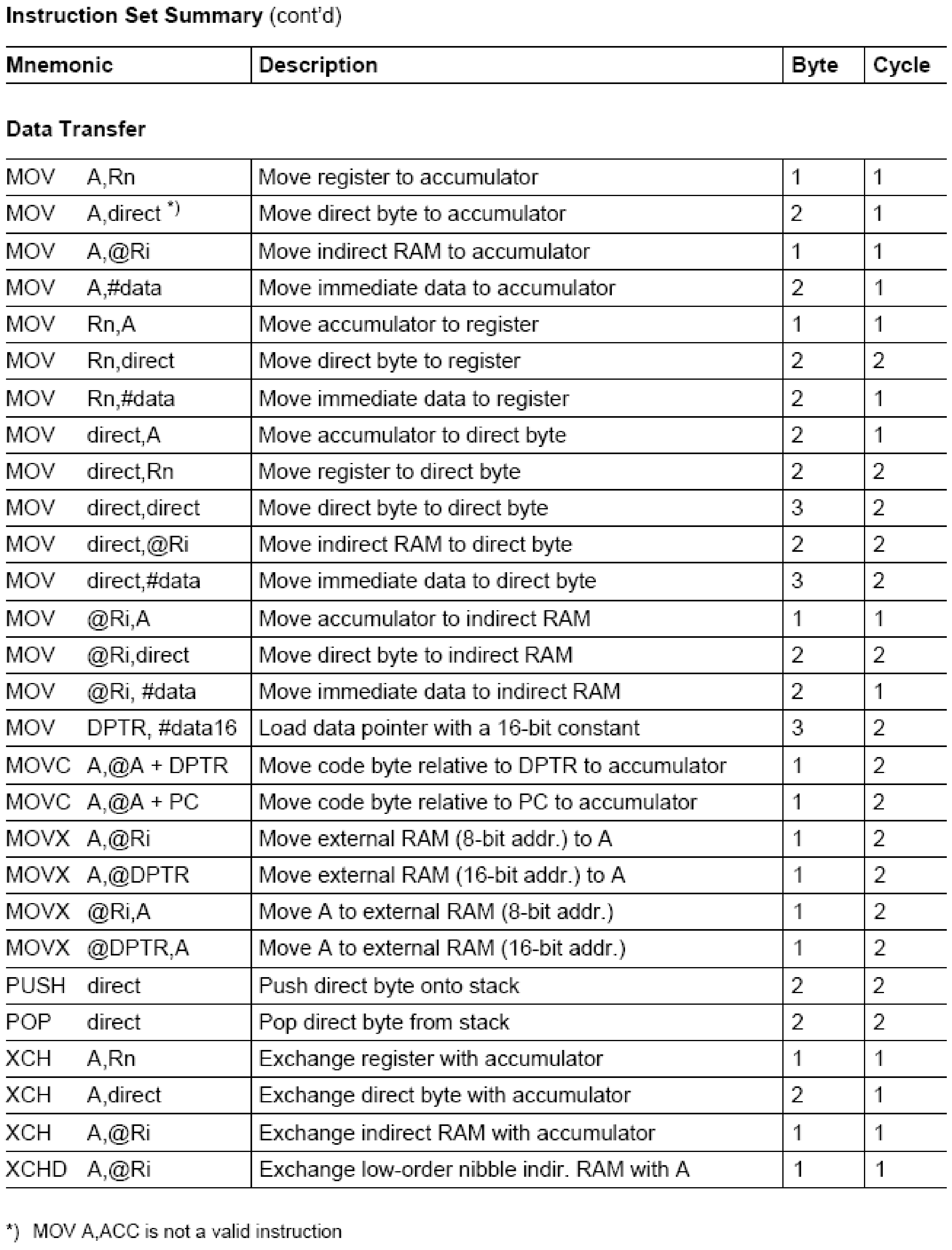
Instruction Set Summary

Mnemonic	Description	Byte	Cycle
Arithmetic Operations			
ADD A,Rn	Add register to accumulator	1	1
ADD A,direct	Add direct byte to accumulator	2	1
ADD A, @Ri	Add indirect RAM to accumulator	1	1
ADD A,#data	Add immediate data to accumulator	2	1
ADDC A,Rn	Add register to accumulator with carry flag	1	1
ADDC A,direct	Add direct byte to A with carry flag	2	1
ADDC A, @Ri	Add indirect RAM to A with carry flag	1	1
ADDC A, #data	Add immediate data to A with carry flag	2	1
SUBB A,Rn	Subtract register from A with borrow	1	1
SUBB A,direct	Subtract direct byte from A with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB A,#data	Subtract immediate data from A with borrow	2	1
INC A	Increment accumulator	1	1
INC Rn	Increment register	1	1
INC direct	Increment direct byte	2	1
INC @Ri	Increment indirect RAM	1	1
DEC A	Decrement accumulator	1	1
DEC Rn	Decrement register	1	1
DEC direct	Decrement direct byte	2	1
DEC @Ri	Decrement indirect RAM	1	1
INC DPTR	Increment data pointer	1	2
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal adjust accumulator	1	1



Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Logic Operations			
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	1
ANL A,@Ri	AND indirect RAM to accumulator	1	1
ANL A,#data	AND immediate data to accumulator	2	1
ANL direct,A	AND accumulator to direct byte	2	1
ANL direct,#data	AND immediate data to direct byte	3	2
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	1
ORL A,@Ri	OR indirect RAM to accumulator	1	1
ORL A,#data	OR immediate data to accumulator	2	1
ORL direct,A	OR accumulator to direct byte	2	1
ORL direct,#data	OR immediate data to direct byte	3	2
XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A direct	Exclusive OR direct byte to accumulator	2	1
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL A,#data	Exclusive OR immediate data to accumulator	2	1
XRL direct,A	Exclusive OR accumulator to direct byte	2	1
XRL direct,#data	Exclusive OR immediate data to direct byte	3	2
CLR A	Clear accumulator	1	1
CPL A	Complement accumulator	1	1
RL A	Rotate accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within the accumulator	1	1



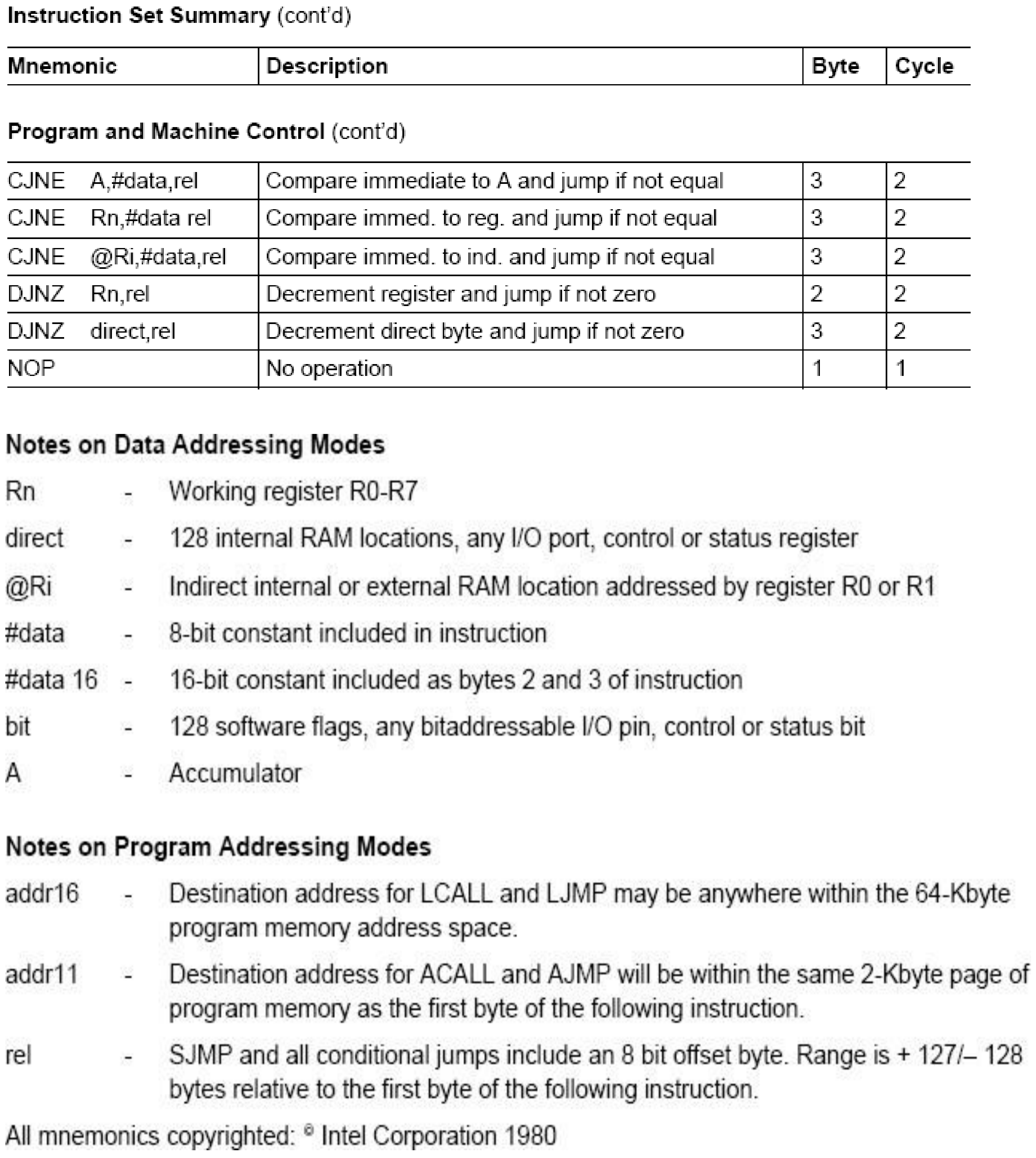
Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Data Transfer			
MOV A,Rn	Move register to accumulator	1	1
MOV A,direct *)	Move direct byte to accumulator	2	1
MOV A,@Ri	Move indirect RAM to accumulator	1	1
MOV A,#data	Move immediate data to accumulator	2	1
MOV Rn,A	Move accumulator to register	1	1
MOV Rn,direct	Move direct byte to register	2	2
MOV Rn,#data	Move immediate data to register	2	1
MOV direct,A	Move accumulator to direct byte	2	1
MOV direct,Rn	Move register to direct byte	2	2
MOV direct,direct	Move direct byte to direct byte	3	2
MOV direct,@Ri	Move indirect RAM to direct byte	2	2
MOV direct,#data	Move immediate data to direct byte	3	2
MOV @Ri,A	Move accumulator to indirect RAM	1	1
MOV @Ri,direct	Move direct byte to indirect RAM	2	2
MOV @Ri, #data	Move immediate data to indirect RAM	2	1
MOV DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOVC A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX @Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX @DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH direct	Push direct byte onto stack	2	2
POP direct	Pop direct byte from stack	2	2
XCH A,Rn	Exchange register with accumulator	1	1
XCH A,direct	Exchange direct byte with accumulator	2	1
XCH A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

*) MOV A,ACC is not a valid instruction

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Boolean Variable Manipulation			
CLR C	Clear carry flag	1	1
CLR bit	Clear direct bit	2	1
SETB C	Set carry flag	1	1
SETB bit	Set direct bit	2	1
CPL C	Complement carry flag	1	1
CPL bit	Complement direct bit	2	1
ANL C,bit	AND direct bit to carry flag	2	2
ANL C,/bit	AND complement of direct bit to carry	2	2
ORL C,bit	OR direct bit to carry flag	2	2
ORL C,/bit	OR complement of direct bit to carry	2	2
MOV C,bit	Move direct bit to carry flag	2	1
MOV bit,C	Move carry flag to direct bit	2	2
Program and Machine Control			
ACALL addr11	Absolute subroutine call	2	2
LCALL addr16	Long subroutine call	3	2
RET	Return from subroutine	1	2
RETI	Return from interrupt	1	2
AJMP addr11	Absolute jump	2	2
LJMP addr16	Long iump	3	2
SJMP rel	Short jump (relative addr.)	2	2
JMP @A + DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if accumulator is zero	2	2
JNZ rel	Jump if accumulator is not zero	2	2
JC rel	Jump if carry flag is set	2	2
JNC rel	Jump if carry flag is not set	2	2
JB bit,rel	Jump if direct bit is set	3	2
JNB bit,rel	Jump if direct bit is not set	3	2
JBC bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE A,direct,rel	Compare direct byte to A and jump if not equal	3	2



Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Program and Machine Control (cont'd)			
CJNE A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE Rn,#data rel	Compare immed. to reg. and jump if not equal	3	2
CJNE @Ri,#data,rel	Compare immed. to ind. and jump if not equal	3	2
DJNZ Rn,rel	Decrement register and jump if not zero	2	2
DJNZ direct,rel	Decrement direct byte and jump if not zero	3	2
NOP	No operation	1	1

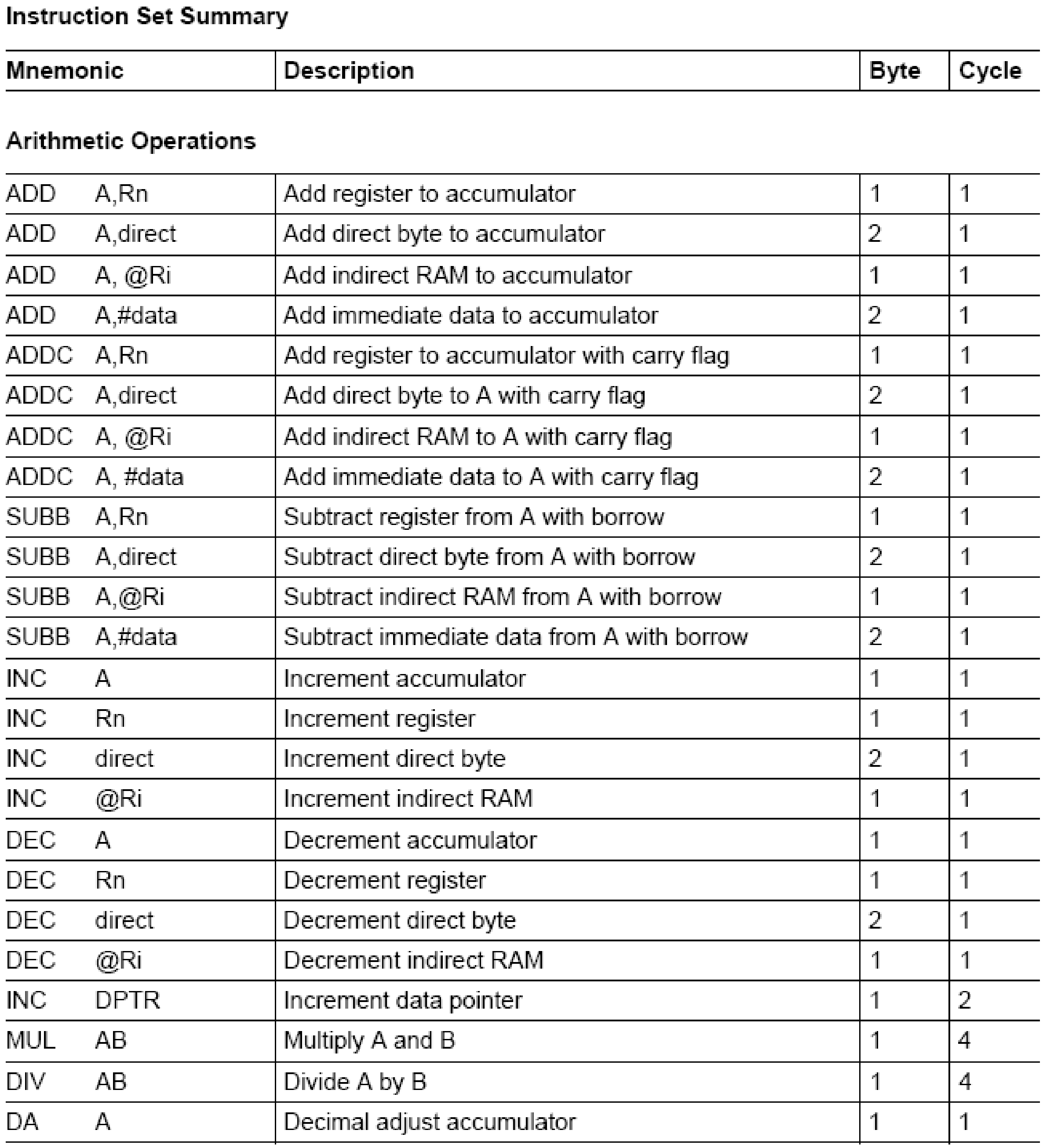
Notes on Data Addressing Modes

Rn	-	Working register R0-R7
direct	-	128 internal RAM locations, any I/O port, control or status register
@Ri	-	Indirect internal or external RAM location addressed by register R0 or R1
#data	-	8-bit constant included in instruction
#data 16	-	16-bit constant included as bytes 2 and 3 of instruction
bit	-	128 software flags, any bitaddressable I/O pin, control or status bit
A	-	Accumulator

Notes on Program Addressing Modes

addr16	-	Destination address for LCALL and LJMP may be anywhere within the 64-Kbyte program memory address space.
addr11	-	Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.
rel	-	SJMP and all conditional jumps include an 8 bit offset byte. Range is + 127/– 128 bytes relative to the first byte of the following instruction.

All mnemonics copyrighted: © Intel Corporation 1980



Instruction Set Summary

Mnemonic	Description	Byte	Cycle
Arithmetic Operations			
ADD A,Rn	Add register to accumulator	1	1
ADD A,direct	Add direct byte to accumulator	2	1
ADD A, @Ri	Add indirect RAM to accumulator	1	1
ADD A,#data	Add immediate data to accumulator	2	1
ADDC A,Rn	Add register to accumulator with carry flag	1	1
ADDC A,direct	Add direct byte to A with carry flag	2	1
ADDC A, @Ri	Add indirect RAM to A with carry flag	1	1
ADDC A, #data	Add immediate data to A with carry flag	2	1
SUBB A,Rn	Subtract register from A with borrow	1	1
SUBB A,direct	Subtract direct byte from A with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB A,#data	Subtract immediate data from A with borrow	2	1
INC A	Increment accumulator	1	1
INC Rn	Increment register	1	1
INC direct	Increment direct byte	2	1
INC @Ri	Increment indirect RAM	1	1
DEC A	Decrement accumulator	1	1
DEC Rn	Decrement register	1	1
DEC direct	Decrement direct byte	2	1
DEC @Ri	Decrement indirect RAM	1	1
INC DPTR	Increment data pointer	1	2
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal adjust accumulator	1	1