UNIT 3

USE MICRO COMPILERS TO CREATE 8051 MICROCONTROLLERS PROGRAMS

Introduction to Micro Compilers and 8051 Programming

Overview

Microcontrollers are small programmable electronic devices used in embedded systems to control machines, devices, and processes. The 8051 microcontroller is one of the most widely used microcontrollers in industrial and educational applications because of its simple architecture and ease of programming.

To make the 8051 perform tasks such as blinking LEDs, controlling motors, reading sensors, and communicating with other devices, programs must be written and converted into machine code. This conversion is done by software called a micro compiler or assembler.

In this unit, students learn how to use micro compilers to create, assemble, test, and run 8051 microcontroller programs. The unit also explains the instruction set of the 8051 and how different instructions are used in practical applications.

Learning Outcomes

By the end of this unit, the learner should be able to:

1. Explain assembling and running an 8051 program.
2. Explain the instruction set of the 8051 microcontroller.
3. Explain data transfer instructions.
4. Explain arithmetic instructions.
5. Explain LOOP, JUMP, and CALL instructions.
6. Develop and test simple 8051 assembly language programs.

---

1. Introduction to Micro Compilers

Meaning of a Micro Compiler

A micro compiler is software used to translate a program written in assembly language or high-level language into machine language that can be executed by a microcontroller.

Examples of commonly used micro compilers and assemblers for the 8051 include:

- Keil uVision
- MIDE-51
- MCU 8051 IDE
- Proteus with assembler support
- ASM51 Assembler

The compiler converts source code into object code or hexadecimal (HEX) files which are loaded into the microcontroller.

A **Hexadecimal file (HEX file)** is a special file that contains the machine code (binary instructions) for a microcontroller such as the **8051**.

It is the final file produced after assembling or compiling a program, and it is the file loaded into the microcontroller memory.

Simple Meaning

When you write a program like this:

```
MOV A,#55H
MOV P1,A
```

the microcontroller cannot directly understand these English-like instructions.

The assembler/compiler converts them into machine code and stores them in a **HEX file**.

Why It Is Called “Hexadecimal”

The file uses **hexadecimal numbers** (base 16 numbers).

Hexadecimal digits are:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Example:

```
55H
0FFH
3AH
```

The letter **H** means the number is hexadecimal.

Example of a HEX File

A HEX file may look like this:

```
:10000000743E75F0FF12001080F5C2D5
:00000001FF
```

This may look confusing, but it contains:

- Memory addresses
- Machine instructions
- Data
- Error checking information

Purpose of a HEX File

The HEX file is used to:

1. Store machine code.
2. Program the microcontroller.
3. Transfer the program from computer to chip.
4. Run the application in the 8051.

Flow of Program Creation

The process is:

```
Assembly Program → Assembler/Compiler → HEX File → Microcontroller
```

Example:

```
Source Code (.asm)
        ↓
Assembler
        ↓
HEX File (.hex)
        ↓
8051 Microcontroller
```

Difference Between ASM File and HEX File

ASM File	HEX File
Human-readable	Machine-readable
Written by programmer	Generated by compiler
Contains assembly instructions	Contains machine code
Extension is .asm	Extension is .hex

Real Example

Suppose you create:

```
MOV A,#55H
```

The assembler converts it into machine language and stores it inside the HEX file.

That HEX file is then loaded into the 8051 chip using a programmer or simulator like:

- Keil uVision
- Proteus
- MCU 8051 IDE

In Short

A HEX file is:

A machine code file generated from a program and used to program the 8051 microcontroller.

Functions of a Micro Compiler

A micro compiler performs the following functions:

1. Converts assembly language into machine code.
2. Detects syntax errors in the program.
3. Produces executable files.

4. Helps in debugging programs.
5. Simulates program operation.
6. Generates HEX files for programming the microcontroller.

Steps in Creating an 8051 Program

The following steps are followed when creating an 8051 program:

Step 1: Write the Source Program

The programmer writes the assembly language instructions using a text editor or IDE.

Example:

```
MOV A,#55H
MOV P1,A
```

Step 2: Save the Program

The file is saved with an extension such as:

.asm

Step 3: Assemble or Compile the Program

The assembler translates the source program into machine code.

Step 4: Correct Errors

If syntax errors are found, the programmer edits and recompiles the program.

Step 5: Generate HEX File

The assembler produces a HEX file which contains machine code.

Step 6: Load the Program into the Microcontroller

The HEX file is programmed into the 8051 microcontroller using a programmer.

Step 7: Run and Test the Program

The microcontroller executes the program and the output is observed.

2. Assembling and Running an 8051 Program

Program Structure of 8051 Assembly Language

An assembly language program generally contains:

1. ORG statement
2. Instructions
3. Labels
4. Comments
5. END statement

Example:

```
ORG 0000H
MOV A,#0FFH
MOV P1,A
END
```

Meaning of Important Statements

ORG

The ORG directive specifies the starting address of the program.

Example:

```
ORG 0000H
```

This means the program starts from memory location 0000H.

END

The END statement shows the end of the program.

Example Program: Blink an LED

```
ORG 0000H
START:
MOV P1,#0FFH
ACALL DELAY
MOV P1,#00H
ACALL DELAY
SJMP START

DELAY:
MOV R0,#255
HERE: DJNZ R0,HERE
RET
END
```

Explanation of the Program

- `MOV P1,#0FFH` sends HIGH signal to Port 1.
- `ACALL DELAY` creates time delay.
- `MOV P1,#00H` sends LOW signal.
- `SJMP START` repeats the process continuously.
- `DJNZ` decreases register value and jumps if not zero.

This program continuously blinks an LED connected to Port 1.

---

3. Instruction Set of 8051 Microcontroller

Meaning of Instruction Set

The instruction set is a collection of commands that the microcontroller understands and executes.

The 8051 instruction set is grouped into:

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions - Change the flow of program execution by jumping, calling, or returning from subroutines.

5. Bit manipulation instructions - Directly operate on individual bits (set, clear, toggle, test, move).

Types of Addressing Modes in 8051

Addressing modes determine how operands are accessed. These are the methods used by an instruction to specify where its data (operand) is located? OR

Addressing modes are the different methods used by the 8051 to locate the data needed by an instruction

1. Immediate Addressing Mode

Data is directly specified in the instruction.

Example:

MOV A,#25H

Clue: The `#` symbol.

- `#25H` means "use the value 25H directly."
- The data itself is inside the instruction.

2. Register Addressing Mode

Data is stored in registers.

Example:

MOV A,R0

Clue: Operand is a register name (`R0–R7`, `A`, `B`).

- Data is taken from register R1.

3. Direct Addressing Mode

The memory address is directly specified.

Example:

MOV A,30H

Clue: A memory address is written directly, with no `#` and no register name.

- `30H` is treated as a RAM address.
- The 8051 goes to memory location 30H and reads the data stored there.

4. Indirect Addressing Mode

A register contains the address of data.

Example:

MOV A,@R0

Clue: The `@` symbol.

- R0 contains an address.
- The 8051 first looks at R0, then goes to that memory address.

5. Indexed Addressing Mode

The clue for **Indexed Addressing Mode** is that it uses **A (Accumulator) together with DPTR or PC** to calculate the address. Used for accessing lookup tables.

Clues to recognize Indexed Addressing

✓ Contains **`MOVC`** instruction (Move Code byte)

✓ Contains **`@A+DPTR`** or **`@A+PC`**

✓ Address is calculated as:

Base Address (DPTR or PC) + Index (A)

Example:

MOVC A,@A+DPTR

Quick Summary

Example	Addressing Mode	Clue
`MOV A,#25H`	Immediate	`#`
`MOV A,R1`	Register	Register name
`MOV A,30H`	Direct	Memory address written directly
`MOV A,@R0`	Register Indirect	`@`
`MOVC A,@A+DPTR`	Indexed	`@A+DPTR` or `@A+PC`

Memory trick:
`Indexed = Base Address + Index`, so whenever you see **A added to DPTR or PC**, think **Indexed Addressing Mode**.

Note:

DPTR (Data Pointer Register) is a **16-bit register** used to store an address in memory.

PC (Program Counter) is a **16-bit register** that automatically holds the address of the next instruction to be executed.

4. Data Transfer Instructions

Meaning

Data transfer instructions are used to move data from one location to another inside the microcontroller.

These instructions do not change the original data value.

Common Data Transfer Instructions

Instruction	Meaning
MOV	Move data
PUSH	Push data into stack
POP	Remove data from stack
XCH	Exchange data
MOVX	Move external memory data

MOVC Move code memory data

MOV Instruction

The MOV instruction copies data from source to destination.

Example 1

MOV A,#45H

This loads hexadecimal value 45H into accumulator A.

Example 2

MOV R1,A

Copies accumulator content into register R1.

PUSH Instruction

Stores data temporarily into the stack.

Example:

PUSH 0E0H

Pushes accumulator content into stack memory.

POP Instruction

Retrieves data from the stack.

Example:

POP 0E0H

Restores accumulator content from stack.

XCH Instruction

Exchanges data between accumulator and another register.

Example:

XCH A,R0

Accumulator exchanges data with R0.

Example Program Using Data Transfer Instructions

```
ORG 0000H
MOV A,#25H
MOV R0,A
MOV 30H,A
END
```

Explanation

- 25H is loaded into accumulator.
- Data is copied into register R0.
- Data is also stored at RAM location 30H.

---

5. Arithmetic Instructions

Meaning

Arithmetic instructions perform mathematical operations such as addition, subtraction, multiplication, and division.

Common Arithmetic Instructions

Instruction	Meaning
ADD	Addition
ADDC	Addition with carry

SUBB	Subtraction with borrow
INC	Increment
DEC	Decrement
MUL	Multiply
DIV	Divide

ADD Instruction

Adds two numbers.

Example:

```
MOV A,#20H
ADD A,#10H
```

Result:

A = 30H

ADDC Instruction

Adds numbers together including carry bit.

Example:

```
ADDC A,R1
```

Used in multi-byte addition.

`ADDC A, R1` means:

Add the contents of R1 and the Carry flag to A (Accumulator).

Formula

```
A = A + R1 + Carry
```

Example

Suppose:

```
A = 20H
R1 = 10H
Carry = 1
```

After:

```
ADDC A, R1
```

Result:

```
A = 20H + 10H + 1 = 31H
```

SUBB Instruction

Subtracts one number from another including borrow.

Example:

SUBB A,#05H

INC Instruction

Increases a value by 1.

Example:

INC A

DEC Instruction

Decreases a value by 1.

Example:

DEC R0

MUL Instruction

Multiplies two unsigned numbers.

Example:

```
MOV A,#05H
MOV B,#02H
MUL AB
```

Result:

A = 0AH

DIV Instruction

Divides accumulator by register B.

Example:

```
MOV A,#20H
MOV B,#04H
DIV AB
```

Example Program for Addition

```
ORG 0000H
MOV A,#15H
MOV R0,#20H
ADD A,R0
MOV P1,A
END
```

Explanation

- 15H is loaded into accumulator.
- 20H is loaded into register R0.
- Both numbers are added.
- Result is sent to Port 1.

---

6. LOOP Instructions

Meaning of LOOPING

Looping means repeating a group of instructions several times until a condition is satisfied.

Loop instructions reduce program size and execution time.

DJNZ Instruction

DJNZ means “Decrement and Jump if Not Zero.”

Example:

```
MOV R0,#10
HERE: DJNZ R0,HERE
```

Explanation

- R0 is initialized with value 10.
- Each time DJNZ executes, R0 decreases by 1.
- The loop continues until R0 becomes zero.

Example Delay Program

```
ORG 0000H
MOV R2,#200
LOOP1: DJNZ R2,LOOP1
END
```

This creates a simple time delay.

---

7. JUMP Instructions

Meaning

Jump instructions transfer program execution from one memory location to another.

They are used for decision making and looping.

Types of Jump Instructions

Instruction	Meaning
SJMP	Short jump
LJMP	Long jump
AJMP	Absolute jump
JZ	Jump if zero
JNZ	Jump if not zero
JC	Jump if carry
JNC	Jump if no carry

SJMP Instruction

Used for short-distance jumps.

Example:

SJMP START

JZ Instruction

Jumps if accumulator becomes zero.

Example:

JZ NEXT

JNZ Instruction

Jumps if accumulator is not zero.

Example:

JNZ AGAIN

Example Program Using Jump Instructions

```
ORG 0000H
MOV A,#00H
JZ SKIP
MOV P1,#0FFH
SKIP:
END
```

Explanation

- Accumulator is loaded with zero.
- JZ checks the accumulator.
- Since accumulator is zero, program jumps to SKIP.

---

8. CALL Instructions

Meaning

CALL instructions are used to call subroutines.

A subroutine is a small program section that performs a specific task.

After execution, the program returns to the main program.

Types of CALL Instructions

Instruction	Meaning
ACALL	Absolute call
LCALL	Long call

ACALL Instruction

Calls a nearby subroutine.

Example:

ACALL DELAY

RET Instruction

RET returns execution back to the main program.

Example:

RET

Example Program Using CALL Instruction

```
ORG 0000H
START:
MOV P1,#0FFH
ACALL DELAY
MOV P1,#00H
ACALL DELAY
SJMP START

DELAY:
MOV R3,#255
HERE: DJNZ R3,HERE
RET
END
```

Advantages of Using Subroutines

1. Reduces program size.
2. Makes programs easier to understand.
3. Avoids repetition of instructions.
4. Simplifies debugging - Errors can be fixed in one place (the subroutine) instead of many places

---

9. Errors in 8051 Programming

Types of Errors

1. Syntax Errors

These occur when instruction format is incorrect.

Example:

MOV A 25H

Correct form:

MOV A,#25H

---

2. Logical Errors

Program runs but produces wrong results.

Example:

Incorrect jump conditions.

3. Runtime Errors

Occur during execution due to hardware or software faults.

---

1. Simulation and Testing of 8051 Programs

Importance of Simulation

Simulation helps programmers test programs before hardware implementation.

Benefits include:

1. Detecting errors early.
2. Saving time and cost.
3. Observing program operation.
4. Testing hardware connections.

Common Simulation Software:

Proteus

Keil uVision Simulator

MCU 8051 IDE

Steps in Simulation

1. Write the assembly program.
2. Assemble the program.
3. Generate HEX file.
4. Open simulation software.
5. Load the HEX file.
6. Run simulation.
7. Observe outputs.

---

2. Applications of 8051 Programming

The 8051 microcontroller is used in many applications such as:

1. Traffic light control systems.
2. Home automation systems.
3. Industrial automation.
4. Temperature monitoring systems.
5. Security alarm systems.
6. Digital clocks.

7. Motor control systems.
8. Medical electronic devices.

---

3. Advantages of Using Micro Compilers

1. Faster program development.
2. Easier error detection.
3. Improved debugging capability.
4. Simplifies machine code generation.
5. Supports simulation and testing.
6. Increases programmer productivity.

---

4. Precautions During 8051 Programming

1. Use correct syntax.
2. Save programs regularly.
3. Check hardware connections.
4. Use proper power supply.
5. Verify HEX file before loading.
6. Test programs step by step.

Summary

Micro compilers are important tools used to create and test 8051 microcontroller programs. They convert assembly language into machine language that can be executed by the microcontroller. The 8051 instruction set contains different categories of instructions including data transfer, arithmetic, looping, jump, and call instructions.

Understanding how to assemble, debug, simulate, and run programs is important in embedded systems design. The knowledge gained from this unit enables learners to develop practical microcontroller-based applications used in modern electronic systems.

Conclusion

The use of micro compilers in 8051 programming simplifies the development of embedded systems. Through assembly language programming, learners can control microcontroller operations such as data transfer, arithmetic processing, looping, and decision making. Proper understanding of these concepts forms the foundation for advanced embedded systems and automation applications.