1 8051 Assembly

Machine Language set of (binary) strings that are recognized as instructions

Assembly Language symbolic language that can be directly translated into machine language.

Assembler program that translates assembly language into machine language.
1.1 Assembly Language Components

1. instructions expressed as mnemonic and operands, e.g.

   MOV   A, #20

2. assembler directives, e.g.

   ORG   5000h

3. assembler controls
4. comments

   MOV   A, #20 ; Initialize the countdown timer to 20 seconds.

1.1.1 Instructions (labels, mnemonics and operands)

The general form of an assembly instruction is,

```
[label:] mnemonic [operand1] [, operand2] [, operand3] [:comment]
```

where elements in the brackets are not mandatory.

Labels represent code addresses (addresses in program memory), and are used for branching instructions.

Each mnemonic represents a set of (or single) instructions, together with the operands it uniquely represents a single machine instruction or opcode. For example,

MOV   R0, #2
MOV   R1, #2

Although the mnemonic is the same (MOV), these are two different 8051 instructions, since when referencing registers using their name (R0-R7, A, B, DPTR) there are specific machine instructions for each (the last 3 bits of the MOV machine language instruction indicate the register).

The operands in an assembly instructions are seemingly in reverse order, for example with the MOV instruction (more appropriately named copy) and ADD,

```
MOV   <destination>, <source>
ADD   <destination, op1>, <op2>
```

1.1.2 Assembler directives

Assembler directives are mnemonics that are never translated into machine instructions, they are used to direct the assembler program. For example ORG sets the start address (origin) of the program in program memory.

1.1.3 Assembler controls

Assembler controls establish the format of the program listing, and the produced object file, for the most part not having an effect on the assembly program itself, for example INCLUDE(file) and NOERRORPRINT.
1.1.4 Comments

The most important of all are comments! Without comments an assembly program is at best cryptic, at worst incomprehensible to humans. You must learn to comment almost every line of your assembly programs in order to instruct your TA as to what you are trying to accomplish. An example of a helpful, and not at all helpful comment:

ADD A, #1 ; increment the widget counter
ADD A, #1 ; add one to the accumulator

Assembly comments are single line, and are placed on the same line after an instruction, although you can wrap them to multiple lines like so

ADD A, #1 ; increment the super special
; widget counter

With poor commenting even the best assembly program assignment is not likely to get a passing grade. Comments also take on a whole new meaning in exams, where it is often harder to create a syntactically correct program.

2 8051 Microcontroller Overview

![8051 Block Diagram](image)

Figure 1: 8051 Block Diagram[2] (©Philips Semiconductors)

3 8051 Memory Organization

The 8051 microcontroller has a variety of different memory spaces used for different purposes which vary by features and accessibility. They can be broadly divided into two major categories; program
memory and data memory, both of which have internal and external components.

Figure 2: 8051 Memory Structure[2] (©Philips Semiconductors)

### 3.1 Program Memory

Program memory is read-only\(^1\) and has a limit of 64K, with the lower 4K optionally (if \(EA = 1\)) being internal to the microcontroller (internal program memory). On powering up the 8051 begins executing code at address 0\(h\) in program memory.

**External code memory** In the vast majority of applications 4K of code memory is not enough, and often a ROM, EPROM or Flash chip is interfaced to the 8051 to provide a more reasonable amount of program memory, addressable by up to 16 bits (FFFF\(h\)).

### 3.2 Data Memory

Data memory is generally read-write, and also has a limit of 64K. Data memory is much more complex, mainly due to the 8051’s extensive use of multiple register banks, memory mapped IO and bit-addressable memory.

---

\(^1\) program memory is always read-only to the microcontroller, however many special educational and development kits which contain variants of the 8051 that use flash memory allow the user to write to program memory from an external source via the microcontroller’s IO ports, and this is what we will in general be using.
3.3 Internal Data Memory (RAM)

The 8051 has 256 bytes of internal data memory which can be directly addressed, in the lower 128 bytes of internal memory reside the register banks, bit-addressable memory and general purpose ram, while in the higher 128 bytes reside the special function registers.

3.3.1 Register Banks (00 − 1Fh)

Data memory 00 − 1Fh stores the 8051’s 4 register banks each split into 8 registers R0-R7; unlike the typical microprocessor, the 8051 registers can be addressed as internal memory. The 4 banks of registers prove very useful for many applications in which there is frequent context switching, we can avoid pushing the registers on the stack and instead switch register banks.

3.3.2 Bit-Addressable Memory (20 − 2Fh)

Again unlike the typical microprocessor, the 8051 has a region of memory that is both byte and bit-addressable. Bit-addressable memory is often used to store flags/boolean variables, however it is a common mistake to allocate the same location in byte-addressable memory and in bit-addressable memory, and this can be a very difficult problem to debug! You should clearly plan your program’s data allocation to avoid this problem.

3.3.3 General Purpose RAM (30 − 7Fh)

The rest of the first half of internal memory is general byte-addressable memory.
3.3.4 Special Function Registers (SFR) (Direct 80 – FFh)

The second half of internal memory is occupied by the special function registers, which are both bit and byte addressable. These registers are used to perform IO, bit and byte arithmetic, and control many other aspects of the microcontroller. We will look at most of the SFRs in detail by the end of the course, for now the most important to us will be the accumulators A and B, the data pointer DPTR, the Program Status Word (PSW) and the carry bit C contained within.

3.3.5 General Purpose RAM (Indirect 80 – FFh)

Although when directly accessed 80 – FFh contains the SFRs, when indirectly accessed this region is general purpose byte-addressable ram.

3.4 External Data Memory

Because internal memory is so small (128 bytes usable direct, 256 indirect) external data memory (XDATA) is necessarily in most applications. The external memory can only be indirectly accessed however, and is only byte addressable.

4 8051 Addressing Modes.

There are a total of 8 addressing modes used by the 8051 to address data in program and data memory, these are

1. Immediate
2. Register
3. Direct
4. Indirect
5. Relative
6. Absolute
7. Long
8. Indexed

4.1 Immediate

Used to address constants, has the assembly syntax

\[
\text{MOV A, } \#20
\]

where the literal number is represented by \#<number><radixsuffix> (no suffix indicates decimal). For example, we can write hex and binary constants,

\[
\text{MOV A, } \#2FH
text
\]

\[
\text{MOV A, } \#10001010B
\]
References
