Assembly Language

Instructor: Dmitri A. Gusev

Spring 2007

CSC 120.02: Introduction to Computer Science

Lecture 10, February 27, 2007

Basic Definitions

Assembly language is a low-level programming language in which a mnemonic letter code represents each of the machine-language instructions for a particular computer.

Assembly language is a software tool, a symbolic language that can be directly translated into machine language by a system program called an *assembler*.

The output of an assembler is an *object module* containing the machine language program **and** instructions for the *loader* on where to load the program in the computer’s memory.

Basic Definitions (cont’d)

An assembler that runs on one computer and produces object modules for another is called a *cross assembler*.

Resident assembly language is the one recognized by the *resident assembler* that runs on the manufacturer’s own development systems.

Assembly language programs are usually line-oriented, so that each assembly language statement is contained in a single line with a prescribed format.

Let’s consider a typical assembly language such that each line has four fields (not to be confused with those of object-oriented languages, such as Java) arranged as follows:

LABEL OPCODE OPERANDS COMMENTS

Fields of a Typical Assembly Language: LABEL

The LABEL field is optional. A label is an *identifier* (or *symbol*), i.e., a sequence of letters and digits beginning with a letter. Most assemblers allow identifiers at least six characters long.

Every symbol is assigned a *value* internally. The assembler keeps track of labels and their values by maintaining a *symbol table*. In most cases, the value of a symbol is equal to the *memory address* at which the corresponding instruction or data value is stored. Each symbol may be defined only once in a program, but it may be referenced as often as needed.

Fields of a Typical Assembly Language: OPCODE

The OPCODE field contains the mnemonic of either a *machine instruction* or a *pseudo-operation* or *assembler directive*. Mnemonics may have a *size suffix* (.B for ‘byte’, .W for ‘word’ and the like) to indicate the size of the operands.

The assembler may use a *default size* (usually, one word) if the programmer does not provide a size suffix.

Fields of a Typical Assembly Language: OPERANDS

The OPERANDS field specifies zero or more *operands* separated by commas. An operand is an *expression* consisting of symbols, constants, and operators such as + and -. The simplest expression consists of a single symbol or *constant*.

A decimal constant is denoted by a sequence of digits, and a hexadecimal constant is denoted by a sequence of hexadecimal digits preceded by $. Character constants are surrounded by single quotes (‘A’) and have the corresponding ASCII values.

Fields of a Typical Assembly Language: COMMENTS

The COMMENTS field is ignored by the assembler, but it is essential to good programming! This field should describe the algorithms and data structures used in the program. A line beginning with an asterisk (*) is a full-line comment.

Pseudo-Operations and the PC

Pseudo-operations (or *assembler directives*) tell the assembler how to assemble the program, and may or may not generate instructions or data. Examples of pseudo-operations:

- ORG initializes the *program counter* (*PC*)
- EQU assigns the value of its operand to its label. An EQU statement defines an *assembly-time constant*.
- DC (Define Constant) defines one or more *run-time constants*.
- DS (Define Storage) defines storage for *variables*.

Sample Assembly Language Program

```
*                    Add the first 16 integers. This is a full-line comment.
          ORG        $2000     Start at address 2000 hex
START     CLR.W      SUM       Clear SUM
          MOVE.W     ICNT,D0   Get initial value for CNT (counter)
ALOOP     MOVE.W     D0,CNT    Save the current value of CNT
          ADD.W      SUM,D0    D0=SUM+D0;
          MOVE.W     D0,SUM    Store the current sum in memory
          MOVE.W     CNT,D0    Prepare CNT for updating in D0
          ADD.W      #-1,D0    Decrement D0 by 1
          BNE        ALOOP     Iterate if the counter isn’t zero yet
          JMP        SYSA      Done: Go to operating system
SYSA      EQU        $8008     Operating system return address
CNT       DS.W       1         Reserve one word for CNT
SUM       DS.W       1         Reserve one word for SUM
IVAL      EQU        16        Initial value is 16
ICNT      DC.W       IVAL      Store initial value, a constant
          END        START
```

Addressing Modes

The purpose of addressing modes is to provide an *effective address* for an instruction's operand.

In a *direct addressing* mode, the effective address is taken directly from the instruction, **or** computed by combining a value in the instruction with a value in a register.

In an *indirect* (or *deferred*) *addressing* mode, the address calculation yields the address of a memory location that contains the ultimate effective address, called an *indirect address*.