CS17 (Plan for Studies in High Level Assembly)

Textbook source: Randall Hyde, the Art of Assembly Language

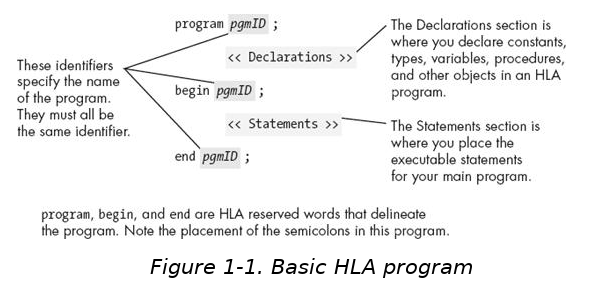
Chapter 1

Goals:

1. Syntax/Structure of HLA programs
2. Intel CPU Architecture introduction (CPU registers)
3. Further language features (control structures, instructions, and data declarations)
4. Standard Library routines
5. Simple program examples

Notes:

Topic 1. Basic Structure of HLA



Further info for Topic 1

* **Section 1.1 Hello World Program**
* #include statements works by informing the HLA compiler to *include* the declarations from a header file(.hhf, HLA Header File), this statement works as a directive informing the compiler *to substitute* the contents of the header file where the #include statement is located
* (***Important Concept***) Compiling an HLA program produces a console application, which when run in a command window (shell) will take control from, execute its instructions, and then return control back to the shell or command-line interpreter.
* More Syntax (highlights only now will be coveredin later chapters)
  + Whitespace ignored = “free format language"
  + String concatenation with no operators (just place side by side)
    - E.g. “Hello ” “World” this is legal and will auto-concatenate
    - E.g. “Hello” nl “World” or with the newline character (nl)
* **Section 1.2 Installation of HLA Compiler**
* **Section 1.3 Declarations and Input/Output Library**
  + basic syntax for variables (static):
    - *identifier* : *type* := *value*;
    - identifier must be a legal name
    - value must be a constant expression, i.e. no variable in RHS of assignment.
    - Declaration without initialization can occur
    - *Identifier* : *type*;
    - More rules:
      * No more than 1 variable declaration per line
      * *variable assignment* should always be on *an uninitialized variable*
  + Input/Output library includes stdin.get() and stdout.put()
    - These functions provide some automated features to convert between string and numeric data types.
    - E.g. Taking input from keyboard value is automatically converted into an integer and stored in a variable as such.
    - E.g. #2 if you specify output of an integer it will be automatically converted to a string.
    - E.g. #3 stdout.put(BoolData) will convert the boolean value to true/false string.
* **Section 1.4 Boolean type**
  + Boolean data type supported (both in language and standard library of routines)
  + Keyword for the argument *type* is boolean
  + Boolean variables fit into a byte objects, which means they are small (1 byte = 8 bits)
  + Also these values can be operated on (see Chapter 2 for discussion on the specific set of machine instruction set for operating on 8-bit values).
* **Section 1.5 Character type**
  + Character objects using a 1-byte character encoding called ASCII for values
  + E.g. static

c: char;

LetterA: char := ‘A’;

* + Character type supported in both input (stdin.get) and output (stdout.put)

*Topic 2 – Machine Instructions*

* **Section 1.6**
  + Intel 80x86 CPU family
    - Von Neumann Architecture Machine: CPU, Memory, and some I/O devices
    - A system bus provides the connections among these, and consists of…
    - Address, data, and control bus
  + The Address bus
    - Used to determine the location of destination to memory location or device ports
    - The ports and memory locations have unique addresses (represented in binary)
  + The data bus
    - Used to pass data between all three components (CPU, Memory, I/O devices)
  + The control bus
    - Controls the direction of flow for data transfer either to or from Mem and I/O device
  + CPU Registers (80x86/Intel)
    - There are 4 types of registers including…a general purpose register:
      * In the Intel architecture it provides the use of 8 general purpose registers for use as well with applications.
      * The names of the 32-bit registers are EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP. The corresponding 16-bit registers are AX, BX, CX, DX, SI, DI, BP and SP. The 8-bit registers are named AL, AH, BL, BH, CL, CH, DL and DH.
      * The purposes of each of the registers is highly specific, so any register cannot be used for just whatever purposes. All the registers have special uses that will limit their use in context.
        + SP/ESP is called the stack pointer, and can’t be used.
        + BP/EBP is also a special purposes register that can’t be used.
      * They are organized in an overlapping way that arranges several of them in a 32-bit space, including 16-bit and 8-bit registers. Because of their overlap they are not all independent, and modifying the larger register will likely have some impact on data in other shared registers within that larger one. (E.g. modifying EAX will often change registers AX and AL and AH. Register value corruption is the effect of what happens when a programmer does not account for the possible problems such as the overlapping registers in the assembly program.
      * EFLAGS is a 32-bit register reserved for kernel mode functions for the operating system, but several of the flags are important to application programming as well.
      * Each value in the 32-bit register is a boolean flag.
      * Of interest to application programmer are the flags for *overflow*, *direction*, *interrupt disable*, *sign*, *zero*, *auxiliary carry*, *parity*, and *carry*.
        + The interrupt flag is mentioned more in Chapter 2
      * Of these eight, the four that are most important are *overflow*, *carry*, *sign*, and *zero*. These make up what are known as condition codes that provide some checks on the results of program state between the operations, e.g. when comparing values. *Parity* is also a condition code, but it will not be covered.
    - An application accessible register (EFLAGS?)
    - A kernel-mode register, which is beyond scope of introductory level class on topic.
    - And a segment register, which is not used by modern operating systems.
      * **NOTE: textbook is geared towards writing programs for 32-bit operating systems**
    - Registers are a middle man in nearly every calculation (addition, subtraction, etc.)