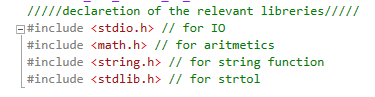
Assembler

The assembler written in C language and its convert the SIMP assembly language into a machine language in which each line contains 5 characters in the base 16.

The input (“asm\_program”) of the assembler is a file that contains a SIMP assembly language and the output (“memin”) is a text file containing the corresponding machine language.

The following are the steps of the assembler:

1. Declaration of the libraries relevant to the Code.



1. Definition of given global variables.

A screenshot of a computer

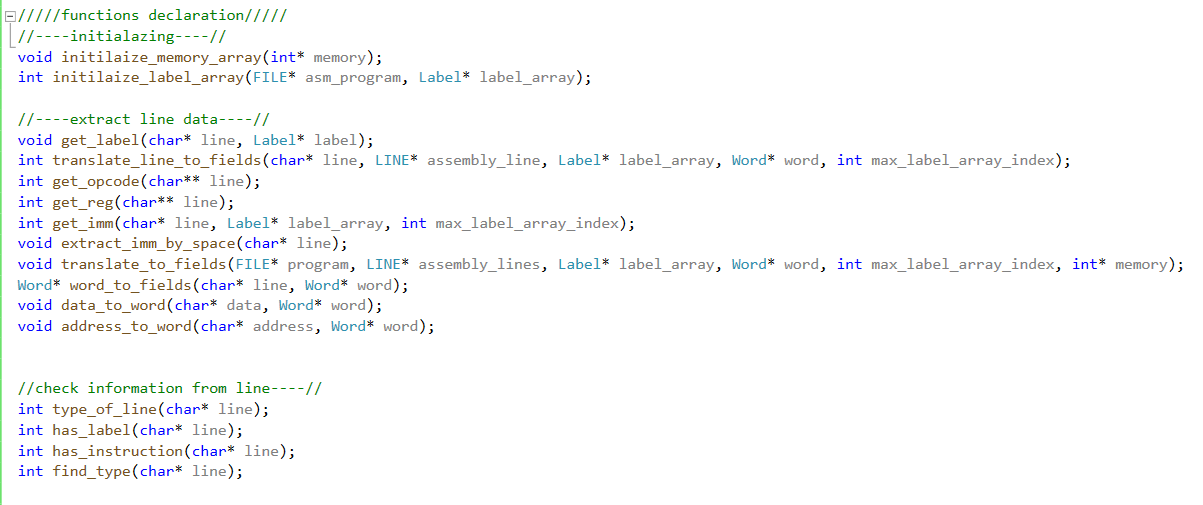
Description automatically generated with medium confidence

1. Definition of 3 data structures, Label, LINE and Word.

Text

Description automatically generated

1. Declaration of functions we will use.

Graphical user interface, text, application

Description automatically generated

1. The main function.

Text

Description automatically generated

* 1. Opening two files and checking that the operation was successful. “asm\_program” for reading and “memin” for writing.
  2. Initialization of major variables:
     1. An array of characters named "label\_array" (Label types) size 4096.
        1. Initialization function: initilaize\_label\_array(FILE\* asm\_program, Label\* label\_array).
        2. Go through the input file “asm\_program” from the beginning till the end, using a while loop. Checks the row type, if the row includes a label, inserting the label and its address into the array. The label address is determined by a variable that is updated each read by row type. Rtype increases by 1, Itype increases by 2, and a label, the ".word" command or empty row do not increase the variable.
        3. Note: we use calloc to allocate memory to this array.
     2. Variable named “word”, a Word Struct.
     3. A 4096-sized array of numbers named “memory” that will contain the rows to be printed to the output file in the appropriate location in the array.
        1. Initialization function: initilaize\_memory\_array(int\* memory).
        2. Insert the string "00000" into every value in the array.
        3. Note: we use calloc to allocate memory to this array.
     4. An array of rows named "assembly\_lines" (LINE types) of size 4096.
        1. Initialization function: translate\_to\_fields(FILE\* program, LINE\* assembly\_lines, Label\* label\_array, Word\* word, int max\_label\_array\_index, int\* memory).
        2. Go through the input file "asm\_program" from beginning till the end using a while loop. Erasing comments from the raw and extract the value of each respective field in the LINE type (opcode, rd, rs, rt, imm). converts the field values to a 5-character base 16 number and save it in the appropriate place in the “memory” array. If the row contains an Itype command, we update the “memory” array with the value of the imm into the appropriate place in the array as well. If the line contains a ".word" command, we insert the data value into the address place in the memory array.
        3. Note: we use calloc to allocate memory to this array.
  3. Writing the text in the output file "memin":
     1. Writing function: write\_output\_to\_file(FILE\* output, int\* memory).
     2. Go through all the “memory” array’s values using a For loop and print them in the output file "memin" line by line corresponding to their location in the array.
  4. Closing the files:
     1. Closing function: close\_files(FILE\* asm\_program, FILE\* memin).
     2. Closes both files, input “asm\_program” and input “memin”.
  5. Free memory allocation:
     1. Free memory function: free\_allocate\_memorry(Label\* label\_array, LINE\* assembly\_lines, int\* memory)
     2. Free the memory allocation of the label array “label\_array”, the array of the struct LINE “assembly\_lines” and the array of int “memory”.

Functions’ explanations:

1. Initialize functions:
   1. void initilaize\_memory\_array(int\* memory);

initialize the memory array as “00000” in each of the 4096 rows.

* 1. int initilaize\_label\_array(FILE\* asm\_program, Label\* label\_array);

initialize the label array, array of labels. Every variable in the array combines the label name and its address.

1. extract line data functions:
   1. void get\_label(char\* line, Label\* label);

extract the label from the line.

* 1. int translate\_line\_to\_fields(char\* line, LINE\* assembly\_line, Label\* label\_array, Word\* word, int max\_label\_array\_index);

extract the label, registers and immediate value to a Line struct.

* 1. int get\_opcode(char\*\* line);

extract the opcode from the line.

* 1. int get\_reg(char\*\* line);

extract the register from the line.

* 1. int get\_imm(char\* line, Label\* label\_array, int max\_label\_array\_index);

extract the imm from the line.

1. Extract line data functions:
   1. void extract\_imm\_by\_space(char\* line);

erase spaces from the imm to get the imm without the spaces.

* 1. void translate\_to\_fields(FILE\* program, LINE\* assembly\_lines, Label\* label\_array, Word\* word, int max\_label\_array\_index, int\* memory);

extract all the fields (label, rd register, rt register, rs register and imm) from the line to the memory array.

* 1. Word\* word\_to\_fields(char\* line, Word\* word);

Extract the address and the data of the “.word” command from the line.

* 1. void data\_to\_word(char\* data, Word\* word);

extract the data from the “.word” command from the line.

* 1. void address\_to\_word(char\* address, Word\* word);

extract the address from the “.word” command from the line.

1. check information from line functions:
   1. int type\_of\_line(char\* line);

check the type of the line. 0 for empty row, 1 for only label, 2 for label + Rtype, 3 for label + Itype, 4 for only Rtype, 5 for only Itype, 6 for .word

* 1. int has\_label(char\* line);

check if there is a label in the line.

* 1. int has\_instruction(char\* line);

check if there is an Rtype or Itype instruction in the line.

* 1. int find\_type(char\* line);

check if the line is with Itype (return 2) or Rtype (return 1).

1. Erase spaces and hashtags functions:
   1. char\* skip\_seperator(char\* line);

erase separators such as spaces and “,” from the line.

* 1. void erase\_hashtag(char\* line);

erase “#” from the line

* 1. char\* erase\_spaces\_in\_the\_end(char\* line);

erase spaces from the end of the line.

1. writing functions:
   1. int hex\_value(LINE\* assembly\_lines);

combine all the fields values to one base 16 value with 5 digits.

* 1. void write\_output\_to\_file(FILE\* output, int\* memory);

write line by line the values from the memory array to the output file according to the index.

* 1. void close\_files(FILE\* asm\_program, FILE\* memin);

function that gets 2 pointers to files and close them.

* 1. void free\_allocate\_memorry(Label\* label\_array,LINE\* assembly\_lines, int\* memory);

function that gest 3 pointers and free the memory allocated for creating them.

1. base number exchange functions:
   1. void address\_hexe\_to\_int(char\* address, int address\_int);

Convert the 16 base char to 10 base int.

* 1. void base10\_to\_base16(char\* base10, char\* base16);

Convert base 10 char to base 16 char.

* 1. char int\_base16\_to\_char\_base16(int base16);

Convert 16 base int to 16 base char.

* 1. void char\_16base\_to\_char\_16base\_16scomp(char\* base16);

Convert 16 base char to 16 base char in 16s comp. meaning to the negative representation.

* 1. int char\_base16\_to\_int\_base10(char d);

convert 16 base char to 10 base int.

* 1. int base10\_char\_to\_base16\_int\_16scomp(char\* base16, char\* base10);

convert 10 base char to 16 base int in 16s comp. meaning to the negative representation.