REPLACED SHEETS SG
☑ Description : paged 1 to paged 38
☑ Claim(s) : paged 39(1) to paged 45
☑ Drawing(s) : paged ..... to paged .....
☑ Abstract : paged 46 to paged -

IPOS's copy!

9801032-5

Graphical Microcontroller Software Development System Flow Chart Engine

This Sheet Has Been Replaced



Field of the Invention

The present invention relates to the methods of a system for the development of assembly language codes for microcontrollers.

Description of the Prior Art

Flow charts have been used to pictorially represent program flow in software systems. They are useful because they help the programmer in visualising and comprehending the software structure and flow. This is especially important in the case of assembly language programs, which are generally more difficult to understand and troubleshoot than other more high-level programming languages.

There are existing computer-aided software engineering (CASE) tools that are capable of generating flow charts from assembly language codes. These flow charts consist of shapes that are directly mapped from the assembly instructions. The flow charts created are static and cannot be edited. Subsequent conversion back to the source codes is not possible. Therefore, the flow chart only serves to provide a graphical representation of the program. If the user encounters a software error within the flow chart, the necessary correction can only be made in the assembly code text.

On the other hand, CASE tools are available to generate assembly language codes from flow diagrams. The program is constructed by creating flow diagrams with special symbols, followed by the addition of assembly instructions into each symbol. The fully constructed diagram is then converted into assembly language

codes. When the programmer discovers errors in the codes, changes have to be made to the diagram, rather than the code. The programmer has to locate the exact symbol or flow structure in the diagram to make the amendments. This is both tedious and cumbersome. In addition, the tools are incapable of converting the assembly codes back into diagrams.

There is therefore a need for a CASE tool that can handle both the creation of editable flow charts from assembly codes and also the generation of assembly codes from its flow chart. It must be flexible enough to manage real world assembly language programs that are constructed using unconventional and unstructured programming practices that often result in convoluted code that is difficult to comprehend and troubleshoot. Such tools would have to employ intelligent and robust algorithms in the conversion operations. They must ensure that the codes and the flow charts bear exact correlation with each other. In addition, the facility to modify the flow charts requires special graphical editing techniques.

Summary of the Invention

In accordance with one embodiment of the present invention there is disclosed a graphical microcontroller software development system flow chart engine, said engine comprising:

a flow chart navigator engine sub-system;

an assembly code to flow chart conversion engine sub-system;

a flow chart to assembly code conversion engine sub-system; and

a flow chart editing engine sub-system.

The flow chart navigator engine sub-system comprises:

a tree branch expander-contractor sub-system;

a tree parent node state setter sub-system;

a tree branch adder sub-system;

This Sheet Has Been Replaced



a tree refresher sub-system; and

a tree node selection handler sub-system.

The flow chart navigator engine sub-system consists of a means to provide a hierarchical view of the entire flow chart in a tree structure format, a means to allow branches in the tree to be selectively expanded and contracted by the user and a means to automatically update the tree structure whenever a change in the flow chart occurs such that the tree and the flow chart are always synchronised.

The tree branch expander-contractor sub-system consists of a means to manage the expansion and contraction of tree branches to allow the lower level contents of parent nodes to be selectively revealed or hidden.

The tree parent node state setter sub-system consists of a means to track the status of parent nodes, which may either be in the expanded or contracted state, a means to search for all flow chart shapes that are parents and to set the states of their corresponding parent nodes in the tree accordingly and a means to identify the tree's associated flow chart editing window so that shape information contained in the window can be retrieved.

The tree branch adder sub-system consists of a means to insert parent nodes into the tree, a means to identify the tree's associated flow chart editing window for the retrieval of shape information and a means to specifically add child nodes for a particular parent node.

The tree refresher sub-system consists of a means to update the tree structure, a means to delete nodes and tree branches, a means to invoke the tree branch adder sub-system to insert new branches into the hierarchical tree and a means to invoke the tree node state setter sub-system to set the expanded or contracted state of parent nodes.

The tree node selection handler sub-system consists of a means to allow quick and easy access of particular flow chart shapes in the flow chart editing window, a means to respond to node selections, a means to specify the flow chart level that is to be displayed, a means to specify the flow chart shape to be highlighted in the flow

This Sheet Has Been Replaced

chart editing window and a means to locate the associated flow chart editing window for the retrieval of flow chart shape data.

The assembly code to flow chart conversion engine sub-system comprises:

a text extractor sub-system;

an assembly instruction database sub-system;

an instruction scanning and storage sub-system;

a function detection sub-system; and

a connecting and branch line creation sub-system.

The assembly code to flow chart conversion engine sub-system consists of a means to convert assembly codes, which can include constructs of a commonly used language, such as the C language, into hierarchical flow charts.

The text extractor sub-system consists of a means to identify the windows containing assembly code and a means to retrieve and manipulate the text strings into a form suitable for instruction scanning.

The assembly instruction database sub-system consists of a means to load the appropriate language instruction set into memory and a means to store the instruction set in a manner that allows the program instructions to be easily categorised.

The instruction scanning and storage sub-system consists of a means to scan and parse the code for labels, instructions, comments and other assembly language constructs and a means to store the created label, instruction and comment objects.

The function detection sub-system consists of a means to detect the presence of functions in the code, a means to assign the functions to different levels of the hierarchical flow chart using pre-defined heuristics and a means to ensure that the generated flow chart will be logical and understandable by using flow chart integrity checking mechanisms.

The connecting and branch line creation sub-system consists of a means to link successive flow chart shapes by inserting connecting lines and a means to connect branch shapes with their target label shapes by inserting branch lines.

This Sheet Has Been Replaced



The flow chart to assembly code conversion engine sub-system comprises:

a connectivity detection sub-system;

a flow chart traversing sub-system; and

a text output sub-system.

The flow chart to assembly code conversion engine sub-system consists of a means to convert the edited hierarchical flow charts back to assembly codes, which can include constructs of a commonly used language, such as the C language.

The connectivity detection sub-system consists of a means to perform a check on the correctness of the flow chart structure and shape contents before the conversion process and a means to check the connectivity between the shapes and lines.

The flow chart traversing sub-system consists of a means to intelligently track the course of the multiple level flow chart and to examine its hierarchical structure for the generation of assembly codes and to also identify and convert certain flow chart structures into their equivalent C language constructs.

The text output sub-system consists of a means to combine the label, instruction and comment information into a line of text, a means to channel the text output to a new file or to overwrite the text in the original code file and a means to automatically indent the text output for increased legibility.

The flow chart editing engine sub-system comprises:

a flow chart drawing sub-system;

a flow chart insertion sub-system;

a text in-place edit sub-system;

a branch line removal sub-system; and

a branch line creation sub-system.

The flow chart editing engine sub-system consists of a means to display and to edit the hierarchical flow chart.

This Sheet Has Been Replaced



The flow chart drawing sub-system consists of a means to draw flow chart shapes and lines onto the flow chart editing window, a means to access the screen co-ordinates, shape type, instruction text and other relevant data of every object in the flow chart database, a means to dynamically adjust the sizes of flow chart shapes to prevent their edges from falling in between the grid squares, a means to automatically orientate the line arrow heads and a means to dynamically change between straight and right-angled lines.

The flow chart insertion sub-system consists of a means to insert segments of flow chart shapes into existing flow charts, a means to provide a text window for text input, a means to scan and parse instructions and subsequently store them into a temporary buffer, a means to detect the presence of functions, a means to compute the dimensions of the inserted flow chart, a means to create connecting and branch lines for the inserted shapes, a means to assign parent indexes to the objects in the inserted flow chart, a means to detect branch lines that traverse across flow chart levels, a means to designate unique indexes to the objects, a means to generate line numbers for every shape, a means to append target label line numbers to branch instructions and a means to check for the existence of single and multiple level recursive calls.

The text in-place edit sub-system consists of a means to allow the user to change the label, instruction or comment text for any flowchart shape, a means to create temporary editable text boxes that are placed over the text concerned and a means to redirect branch lines automatically when the target labels within branch instructions are modified.

The branch line removal sub-system consists of a means to delete outgoing branch lines from a branch shape and a means to identify the branch lines that are connected to a branch shape.

The branch line creation sub-system consists of a means to create branch lines that connect branch shapes with their target label shapes, a means of extracting the target label definition from the branch instruction, a means of locating the target label shape, a means of calculating the dimensions and locations of the new branch lines, a means of assigning the appropriate parent indexes to the branch lines, a means of

This Sheet Has Been Replaced

preventing branch lines from overlapping each other and a means of warning the user if the target label shape does not exist.

Brief Description of Drawings

Fig. 1 is a schematic block diagram of the various flow chart engine sub-systems that are used in the graphical microcontroller software development system, according to an embodiment of the invention;

Fig. 2 is a schematic block diagram illustrating the flow chart navigator engine, showing the interaction and data flow between the various sub-systems;

Fig. 3 is a schematic flow chart depicting the operation of the tree branch expander-contractor sub-system in the flow chart navigator engine;

Fig. 4 is a schematic flow chart showing the operation of the tree parent node state setter sub-system in the flow chart navigator engine;

Fig. 5 is a schematic flow chart of the operation of the tree branch adder sub-system in the flow chart navigator engine;

Fig. 6 is a schematic flow chart illustrating the operation of the tree refresher sub-system in the flow chart navigator engine;

Fig. 7 is a schematic flow chart depicting the operation of the tree node selection handler sub-system in the flow chart navigator engine;

Fig. 8 is a schematic diagram showing the relationship between parent and child objects;

Fig. 9 is a schematic block diagram illustrating the assembly code to flow chart conversion engine, showing the interaction and data flow between the various sub-systems;

Fig. 10 is a schematic flow chart depicting the operation of the text extractor sub-system in the assembly code to flow chart conversion engine;

This Sheet Has Been Replaced



Fig. 11 is a schematic flow chart showing the loading of the language instruction database into memory by the assembly instruction database sub-system in the assembly code to flow chart conversion engine;

Fig. 12 is a schematic flow chart showing the scanning of instructions and the creation of objects in the assembly code to flow chart conversion engine;

Fig. 13 is a schematic block diagram illustrating the flow chart to assembly code conversion engine, showing the interaction and data flow between the various sub-systems;

Fig. 14 is a schematic flow chart of the operation of the connectivity detection sub-system in the flow chart to assembly code conversion engine;

Fig. 15 is a schematic diagram showing the method for checking connectivity between objects;

Fig. 16 is a schematic flow chart depicting the operation of the text output sub-system in the flow chart to assembly code conversion engine;

Fig. 17 is a schematic block diagram illustrating the flow chart editing engine, showing the interaction and data flow between the various sub-systems;

Fig. 18 is a schematic flow chart showing the operation of the flow chart drawing sub-system in the flow chart editing engine;

Fig. 19 is a schematic flow chart of the operation of the text in-place edit sub-system in the flow chart editing engine;

Fig. 20 is a schematic flow chart of the operation of the branch line removal sub-system in the flow chart editing engine;

Fig. 21 is a schematic flow chart of the operation of the branch line creation sub-system in the flow chart editing engine;

Fig. 22 is a schematic diagram showing the change in orientation of a right-angled connecting line using the 'Shift' key;

Fig. 23 is a schematic diagram illustrating flow chart shapes together with their accompanying comments;

Fig. 24 is a schematic diagram showing the in-place text edit box for modifying the text within shapes;

This Sheet Has Been Replaced

Fig. 25 is a schematic diagram showing the state of the branch shape and its accompanying branch line before the text in-place editing operation;

Fig. 26 is a schematic diagram showing the state of the branch shape and its accompanying branch line during the text in-place editing operation;

Fig. 27 is a schematic diagram showing the state of the branch shape and its new branch line after the text in-place editing operation;

Fig. 28 shows a typical branch instruction with its target label and label line number;

Fig. 29 is a schematic flow chart of the flow chart insertion by text operation in the flow chart editing engine;

Fig. 30 is a schematic flow chart of the instruction scanning and object creation sub-system in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 31 is a schematic flow chart of the function detection sub-system in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 32 is a schematic flow chart of the vertical offset calculation sub-system in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 33 is a schematic flow chart of the branch line creation sub-system in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 34 is a schematic flow chart of the parent index assignment sub-system in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 35 is a schematic flow chart of the sub-system which detects branch lines across levels in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 36 is a schematic flow chart of the sub-system that responsible for assigning indexes to objects in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 37 is a schematic flow chart of the line number generation sub-system in the flow chart insertion sub-system of the flow chart editing engine;

Fig. 38 is a schematic flow chart of the branch target line number generation sub-system in the flow chart insertion sub-system of the flow chart editing engine;

This Sheet Has Been Replaced



Fig. 39 is a schematic flow chart of the functions check sub-system in the flow chart insertion sub-system of the flow chart editing engine; and

Fig. 40 is a schematic flow chart of the flow chart insertion by shapes operation in the flow chart editing engine.

Detailed Description of the Best Mode of Performing the Invention

Fig. 1 shows the architecture of the flow chart engine 1 that is used in the graphical microcontroller software development system. The flow chart engine 1 is made up of the following :

a flow chart navigator engine sub-system 100;

an assembly code to flow chart conversion engine sub-system 200;

a flow chart to assembly code conversion engine sub-system 300; and

a flow chart editing engine sub-system 400.

The flow chart navigator engine sub-system 100 comprises a tree branch expander-contractor sub-system 104, a tree parent node state setter sub-system 105, a tree branch adder sub-system 103, a tree refresher sub-system 102 and a tree node selection handler sub-system 101. The said sub-systems function together to provide a hierarchical representation of the entire flow chart in the form of a tree structure 106. The interaction and data flow between the sub-systems is shown in Fig. 2.

The navigator tree 106 is made up of many branches 107. These branches 107 can in turn have sub-branches. Each branch 107 represents one flow chart layer. Branches 107 in the tree can be expanded or contracted as desired to reveal or to hide additional flow chart details in the lower layers. Each branch 107 contains nodes which indicate the collection of flow chart objects that make up a flow chart layer. The maximum number of branch nodes that can be stored is determined by the amount of available system memory. The flow chart navigator engine sub-system 100 is permanently synchronised to its flow chart editing window 305 to provide an up-to-date tree structure 106 whenever changes are made to the flow chart.

This Sheet Has Been Replaced



The tree branch expander-contractor sub-system 104 manages the expansion and contraction of branches 107 in the navigator tree 106. Only the user selected nodes and branches will be shown. The unwanted ones are hidden. Fig. 3 presents a schematic flow chart of the operation of the tree branch expander-contractor sub-system 104.

When the navigator tree 106 is first produced, only the main branch of the tree is displayed. The sub-system 104 achieves this by expanding the main branch and adding the nodes which represent the shapes in the top level of the flow chart. Subsequently, the user can specify the remaining branches that are to be expanded or contracted.

Every parent node 108 that can potentially produce a branch 107 contains a dummy child node. When a parent node 108 is slated for expansion, its dummy child node is first removed. Afterwhich, the tree branch adder sub-system 103 is invoked 116 to create a new branch with new child nodes. This particular sub-system 103 will search for all flow chart objects in the database 10 that are related to the parent node 108. The data obtained from these identified objects will be used to create the child nodes that make up the branch 107. These child nodes are then examined to determine if any of them are parent nodes themselves. The tree branch expander-contractor sub-system 104 is recursively executed again to expand those new parent nodes.

In the contraction of a parent node 108, all the existing child nodes belonging to the branch 107 are deleted, including the sub-branches. When the removal operation is completed, the tree parent node state setter sub-system 105 is invoked 117 to modify the attributes of the parent node 108 to indicate that its branch 107 is in the contracted state. The navigator tree 106 display is updated accordingly.

The tree parent node state setter sub-system 105 is responsible for maintaining certain attributes of the parent flow chart objects that relate to the expanded or contracted state of their corresponding parent nodes 108 in the tree structure 106. These object attributes are referenced in the construction and updating of the navigator tree 106. The flow chart of the operation of the tree parent node state setter sub-system 105 is shown in Fig. 4.

This Sheet Has Been Replaced



The tree parent node state setter sub-system 105 begins operation by first identifying its corresponding flow chart editing window 305. This window 305 contains the database 10 of flow chart objects that are relevant to the tree 106 concerned. Every flow chart window 305 has its own navigator tree 106. A search is conducted to retrieve 122 the parent node's 108 flow chart object from the database 10. When the object is found, its attributes are modified accordingly.

The tree branch adder sub-system 103 handles the insertion of child nodes belonging to a branch 107 into the navigator tree 106. Fig. 5 shows the schematic flow chart of the tree branch adder sub-system 103.

The navigator tree's 106 matching flow chart editing window 305 has to be first identified. The sub-system 103 then proceeds to collect information 121 from all the flow chart objects from the database 10. A search is conducted to identify the flow chart objects that are the children of the expanding parent node 108 object. An additional criteria is imposed to ascertain whether the identified object is a connecting or branch line object. Such objects should not be displayed as nodes in the navigator tree 106. Only tree branch nodes representing flow chart shapes are created 113.

After all the tree branch nodes have been added to the tree 106, another test is conducted to determine whether any of the recently added nodes are parent nodes themselves. If the attribute information within these parent nodes indicate that their branches are in the expanded state, the tree branch expander-contractor sub-system 104, will be activated to produce a new branch. This process repeats itself until all the parent nodes are addressed.

The tree refresher sub-system 102 is executed whenever the flow chart navigator window sub-system 100 requires updating. This occurs when the user changes the flow chart. The modifications include the replacement of flow chart objects, the addition of new objects or the deletion of existing ones and the creation or removal of flow chart layers. The flow chart navigator sub-system 100 is automatically informed 123 of any changes that are made to the flow chart. This ensures that both the data in the navigator tree 106 and the database 10 are always synchronised.

This Sheet Has Been Replaced

In an updating operation, the tree refresher sub-system 102 initially deletes 112 all the parent and child nodes that are currently present in the navigator tree 106. The tree branch adder sub-system 103 is invoked 115 to create and expand all the branches that were previously expanded prior to the flow chart modification. All the necessary parent and child nodes will be added to reconstruct the tree 106. The tree branch adder sub-system 103 automatically calls itself recursively to handle the many layers of sub-branches. In each instance of operation, the branch adder calls 124 the tree branch expander-contractor sub-system 104. The flow chart object information that is used to create the tree 106 is derived from the latest version of the database 10 after the flow chart changes have been made. Fig. 6 shows a flow chart of the operation of the tree refresher sub-system 102.

The tree node selection handler sub-system 101 allows quick and easy access to any flow chart shapes in the flow chart editing window 305 through the use of the navigator tree 106. When a particular node is selected from the tree, the flow chart editing window 305 will automatically refresh itself to display the chosen flow chart object together with its surrounding flow chart objects. The selected flow chart object will always be drawn and highlighted in the centre of the window. If the selected node is located at a level which is different from the current display level in the editing window 305, the flow chart editing sub-system 400 will retrieve those objects at the chosen node's level from its database 10 and display them.

Fig. 7 shows a flow chart of the operation of the tree node selection handler sub-system 101. Whenever the user selects a node from the navigator tree 106, the tree node selection handler sub-system 101 will immediately be started 111. The sub-system 101 will first locate its corresponding flow chart editing window 305. The window's 305 flow chart database 10 is searched to locate the flow chart object which matches the chosen node. Once identified, information regarding the parents of this object is retrieved 120. The tree node selection handler sub-system 101 then issues a message to the flow chart editing sub-system 400 to display those flow chart objects whose parents are similar to those in the chosen node. In other words, the editing window 305 is refreshed to show a new flow chart layer.

This Sheet Has Been Replaced



The assembly code to flow chart conversion engine sub-system 200 converts assembly codes, which can contain constructs of commonly used languages, such as the C language, into hierarchical flow charts. The user initially selects a text file 20 containing the assembly code for input into the system. The codes are then loaded into memory. Internally, the codes are examined 213 for labels, instructions, comments and other relevant data. This information will be used to construct the flow chart. Each line of code is assigned a flow chart shape according to its function. A line of code, which causes the program to branch to another location, would be assigned a different flow chart shape compared to another line of code which does numerical addition. The mapping of the assembly instruction mnemonic its corresponding shape is determined by the assembly language being selected for use. Every C language construct is also converted to its equivalent representation in the flow chart.

The flow chart is constructed in a top down manner. Vertical connecting lines are inserted between the shapes as they are created. Branch lines 423 are added to join branch instruction shapes 421 with their target label shapes 422. Unique line numbers 433 are assigned to every shape based upon their sequence in the code. Line numbers 433 and comments 435 that accompany each instruction are conditionally drawn depending upon the user preference settings. Using certain pre-defined heuristics, logical blocks of code are grouped into functions, which may in turn be further grouped into sub-functions. Each successive level of program detail constitutes a layer. This layered approach gives the user a hierarchical overview of the entire program. Therefore, at the upper levels, a particular flow chart shape may represent a collection of codes beneath it. The different levels of the flow chart may be accessed with the help of the flow chart navigator engine sub-system 100.

Assembly code integrity checking mechanisms are installed into the conversion engine sub-system 200. These are used to ensure that the flow chart created will be logical and understandable. Program abnormalities such as instances of missing or duplicate labels, convoluted code and calls without returns, amongst others, are highlighted to the programmer.

The assembly code to flow chart conversion engine sub-system 200 comprises a text extractor sub-system 201, an assembly instruction database sub-system 202, an

This Sheet Has Been Replaced



instruction scanning and storage sub-system 204, a function detection sub-system 203 and a connecting and branch line creation sub-system 205. The interactions and data flow between these sub-systems is shown in Fig. 9.

The system consists of several different types of windows and their associated database. Accessing a specific window requires that all the windows present in the system be examined in turn. At each cycle, a check is made to ascertain if the current window is the one that is desired. If the search is successful, then a pointer is created for that particular window. This pointer will then be used to access the data within the window's database. Pointers to the windows containing the assembly codes are required for the conversion process. The text extractor sub-system 201 utilises the code window 206 pointer to access 223 the individual lines of code in the text files, as depicted in the flow chart in Fig. 10. The contents of one text file 20 is loaded into one code window 206. If there is more than one text file 20 serving as input, then a new code window 206 pointer is created for every file 20. One flow chart is created for every text file.

The assembly instruction database sub-system 202 enables the instruction scanning and storage sub-system 202 to correctly interpret the assembly instructions and syntax. The system first determines the language that is chosen for the interpretation of the assembly codes. The appropriate instruction set is then loaded from the corresponding language database file. Fig. 11 shows the method to load the instruction set into program memory 207. Each instruction is stored in an instruction object class. These instruction objects are in turn stored in an instruction object array 207. The instruction object class contains two members. One member stores the instruction name while the other is used to indicate the category of the instruction. One object is created for every instruction loaded. The object array stores pointers to the instruction objects. The array can dynamically shrink and grow as necessary. The pointers are referenced by an array index.

Labels may be used by assembly programmers to indicate certain strategic points within the program. They also serve to specify the destinations of branch instructions, which direct program flow basing upon certain run-time conditions. The location of every label is needed, so that every branch line that emerges from a branch

This Sheet Has Been Replaced



shape has a destination. An instance of a label object class is created for every label encountered in the code. The label objects are stored in a label object array. They are referenced by an array index. This array will be used by the connecting and branch line creation sub-system 205 to determine the destination for branch lines emerging from branch shapes. The label object class contains two members. One member stores the name of the label while the other stores its location in the flow chart.

Labels are used in three different ways in the assembly code. A line of code might contain only a label, or a label with an instruction, or only an instruction. The system must be able to identify these three situations. In each code window 206, the number of text lines is sought. Leading and trailing tab spaces, including comments, are deleted, leaving only the label and/or instruction. Blank lines are ignored.

Every line is examined in detail. A check is first made to detect the presence of a valid instruction within the line of text with the help of the instruction object array 207 of the assembly instruction database sub-system 202. If an instruction is present. the rest of the line is inspected for the occurrence of a label. On the other hand, if no instruction is present, the text line is assumed to contain only a label. In this case, a label shape object is created and stored in the flow chart database 10.

The method for scanning and storing instruction sequences is almost similar to that for labels. Fig. 12 shows the flow chart of this operation. Once a line of code has been identified, the program would proceed to check the category of the instruction. The instruction text is passed 211 to the assembly instruction database sub-system 202, which then returns 214 the instruction type. Depending on the type of instruction encountered, the appropriate shape would be drawn. Each shape is represented by an instance of the draw object class. This class stores all information relating to the shapes. Before the shapes can be displayed in the flow chart editing window 305, screen co-ordinates have to be assigned to them. The program contains a global screen co-ordinate counter that tracks the position of newly added shapes. The system creates the flow chart in a top down manner. When a new flow chart layer is created, the global counter is reset. The flow chart in each layer begins at the top. Each shape is initially assigned a default size.

This Sheet Has Been Replaced

Whenever a shape is created, an accompanying vertical connecting line with an arrow is generated 212 by the connecting and branch line creation sub-system 205. The co-ordinates of this line is calculated such that it connects the shapes in sequence. A draw object is also created for each connecting line.

As each shape and line is produced, its corresponding draw object is added 221 to the database 10. The shape and line objects are stored as an ordered, non-indexed list of elements implemented as a double linked list. Elements can be inserted or deleted at the head, tail or in the middle of the list. This list is stored in the database 10 that is associated with every flow chart editing window 305. After scanning all the required code files, the end result is a database 10 containing a list of objects, each representing a label, instruction or line. Every draw object is assigned a unique index number as they are created. This number provides a means of identifying the individual objects.

The draw object class members store the shape's or line's screen co-ordinates, their unique index number, and other members that hold the index of the previous and the following shapes or lines connected to it. This is used to track the sequence of the shapes and lines in the flow chart. To track the parent 233 and child 234 relationship between objects at different flow chart layers, a parent index array member is used to store the index of a shape's parent 233. Objects belonging to a certain flow chart level share the same parent index, even though they each have a unique index of their own. The parent index of objects situated at the top most level 231 is zero. The parent index of shapes at the lower layers 232 is simply the index number of their parent shape. These relationships are shown in Fig. 8. An object can have multiple parents. This occurs when the same function is called several times by different parts of the program. The shapes that represent calls to the function would constitute the function's parent.

Flow chart layers are defined by the function detection sub-system 203. This occurs after all the instructions have been scanned and stored. This sub-system 203 uses certain pre-defined rules to intelligently ascertain the structure and organisation of the assembly codes. Each logical block of instructions that work together

This Sheet Has Been Replaced

constitutes a function or a flow chart layer. Additional layers are created if the function can be further subdivided into smaller sections of code.

The parent index array information is also used in the construction of the tree structure 106 in the flow chart navigator window. The image index member of the draw objects stores an integer whose value determines the type of bitmap that is to be drawn beside the node in the tree structure 106. This integer is derived from the shape instruction's category. Another member holds a boolean variable to track whether the branch 107 that belongs to a particular parent node 108 has been expanded or contracted in the tree structure 106.

The label and instruction text data are stored as string members in the draw object. This is the string that appears at the centre of the shape when it is displayed. Another string member holds the comment text that accompanies the instruction in the source code. The comment text is displayed in a separate rectangle to the right of the shape.

Every branch instruction has a destination label 422. Upon encountering a branch instruction, a microcontroller's subsequent execution path would depend on the success or failure of the specified branch condition. In the flow chart, every branch shape will have two lines emerging from it. One line points to the next instruction, while the other line points to the label shape 422 that is referenced by the branch instruction. If the label 422 is non existent, a warning message would be issued to the user.

Whenever a branch instruction is found, a search is done to determine the location of the label 422. The entire text file is first scanned for labels before the scanning of instructions commences. When the label shape 422 belonging to a branch shape 421 is found, a computation is made to determine the distance between the two shapes. This value is used for the construction of the branch line 423.

The flow chart drawing routine accesses each draw object from the flow chart database 10 in turn. Each object's screen co-ordinates, shape type, label, instruction and comment text and other data is extracted and used for drawing the object in the flow chart editing window 305. This proceeds until all the draw objects from the database 10 have been drawn. The final result would be a flow chart editing window

This Sheet Has Been Replaced



305 containing the assembly codes in the hierarchical flow chart format. The window 305 initially displays the flow chart at the top most level. Flow charts at the lower levels are not drawn. They are only drawn when the user opts to examine the flow chart at another level.

The flow chart to assembly code conversion sub-system 300 converts hierarchical flow charts into assembly codes, which can contain constructs of commonly used languages, such as the C language. The user initially selects the flow chart that is targeted for the conversion operation. Internally, the entire flow chart's structure and connectivity between its shapes and lines is determined. This information will be used to generate the codes. Each flow chart shape type represents a language code category. The sub-system 300 tracks the course of the flow chart layer by layer, from the top to the lowest. Depending on the shape encountered during the conversion process, the sub-system 300 takes the appropriate action when generating the codes. The labels, instructions and comments are placed with different offsets from the left margin of the text output. This indentation effect makes the codes more readable. A new line of text is created for every shape. The sequence by which the shapes are examined is determined by their relative positions with respect to their neighbouring connecting lines.

Flow chart integrity checking mechanisms are installed into the conversion sub-system 300. These are used to ensure that the code generated from the flow chart will be both logical and understandable. Missing connecting lines, branch lines, empty levels and other flow chart anomalies are highlighted to the programmer before the conversion commences.

The flow chart to assembly code conversion sub-system 300 comprises a connectivity detection sub-system 303, a flow chart traversing sub-system 301 and a text output sub-system 302. Fig. 13 presents a schematic diagram illustrating the interactions and data flow between the various sub-systems.

Shapes are linked together by connecting lines. The location and orientation of the arrow heads on the lines determine the flow sequence of the shapes. Therefore, it is necessary to determine the connectivity between the lines and the shapes in each

This Sheet Has Been Replaced

flow chart level. The connectivity information thus generated will be used to order the labels, instructions and comments in the assembly code output.

Fig. 14 shows a flow chart of the algorithm to detect connectivity between a shape and its neighbouring line. Shapes and lines are treated equally as draw objects in the algorithm. A particular object's 321 size, as shown in Fig. 15, is first inflated by a small pre-determined amount. This object's 321 newly enlarged co-ordinates will be used as a reference when all the rest of the objects' sizes are inflated in turn. A function is used to check for any overlap 325 in the area covered between the reference object 321 and the rest of the objects. Only objects within the same level are compared. When an overlap is encountered between the enlarged reference object 323 and another inflated object 324, the previous index and next index members of these objects are updated. The previous and next index members indicate the unique index number of the objects preceding and following them in the flow sequence respectively. In this case, the next index of the reference object 321 and the previous index of the its neighbouring object 322 are updated. An error message 313 is produced to notify the user whenever an object is found that is not connected to any other object.

The above process is repeated for all the objects in all the levels. The manner in which the flow chart levels are accessed is determined by the flow chart traversing sub-system 301. When all the levels are exhausted, the previous and next index members of every shape and line would have been assigned a number.

The user is given a choice to channel the generated codes either to the original code file or to a new file 20. In either case, the system has to get a pointer to the appropriate code window 206.

Starting from the first shape in the flow chart, the text output sub-system 302 obtains 310 the line of code associated with the shape and appends it to the code window 206. The text output is indented according to the structure of the code. The shape's next index member will then be used to find its neighbouring line. This process repeats itself until all the shapes' code have been written to the code window 206. The algorithm is graphically represented in the Fig. 16.

This Sheet Has Been Replaced

The process of displaying shapes and lines onto the flow chart editing window 305 occurs after the assembly codes have been converted into their respective flow chart shapes. The database 10 now contains all the draw objects.

The flow chart editing engine sub-system 400 comprises a flow chart drawing sub-system 402, a flow chart insertion sub-system 401, a text in-place edit sub-system 405, a branch line removal sub-system 404 and a branch line creation sub-system 403. The interactions and data flow between the various sub-systems are shown in Fig. 17.

The flow chart drawing sub-system 402 begins operating by accessing 410 every draw object within the database 10 in turn. Each object's screen co-ordinates, shape type, text string and other data are extracted. This information is used to draw the object in the flow chart editing window 305. Initially, only the objects that reside at the upper most level are drawn. Only one flow chart level can be shown at a time. The final result would be a flow chart editing window 305 containing the assembly codes in the flow chart format.

Fig. 18 shows the flow chart of the operation of the flow chart drawing sub-system 402. The instruction text within a shape is only displayed if the shape is large enough to accommodate the whole text string. This is to prevent words from bleeding into the background. The decision to print the text string is automatically handled by the sub-system 402. Therefore, the text can appear and disappear as the user re-sizes the shape.

The program dynamically adjusts the sizes of shapes and lines. The objective is to prevent the edge trackers of the objects from falling in between the grid squares. Edge trackers are the small black squares that are dispersed along strategic points round the border of an object. They are used to re-size the object. A non-branching shape contains 8 edge trackers. 4 of these trackers are situated at the 4 corners of the rectangle, while the remaining at located at the middle points of the shape's sides. If a shape were to be re-sized such that an edge tracker falls in between the grid lines, the connecting and branch lines would not be able to snap to that particular edge tracker. Therefore, both the height and width of the shapes must always span even numbers of grid squares. When the program detects that anyone of the trackers is on the verge of

This Sheet Has Been Replaced

falling in between the grid squares, it will immediately intervene by re-sizing the shape to re-position the affected edge trackers onto the grid boundaries.

The line drawing routine simply plots a line with an arrow head at one end. The direction of the arrow head will always be parallel with the orientation of its end of the line. In addition, the line can alter its shape from that of a straight line to that of a right-angled line dynamically by detecting the movement of the mouse. The orientation of the right-angled line can be changed by pressing the 'Shift' key while dragging the mouse to define the line. The appearance of a right angled connecting line with 431 and without 432 the use of the 'Shift' key is shown in Fig. 22.

In-place editing capability is incorporated into the flow chart editing engine sub-system 400 to facilitate the flow chart shape text editing operation. The in-place edit feature allows the user to change the text contents of the label, instruction and comment data of the shapes. This feature is particularly useful because the user does not have to delete and re-create a flowchart shape each time a correction has to be made to its text. The manner in which text is displayed in shapes is shown in Fig. 23. The comments 435 that accompany an instruction 434 or label are printed in a special rectangular box situated on the right of the shape. Line numbers 433 are situated on the left of the shapes. Text editing only applies to shapes.

Fig. 19 presents a flow chart that illustrates the operation of the text in-place editing sub-system 405. To begin editing, the shape concerned is double-clicked anywhere within its boundaries with the help of a mouse. An editing text box 436 appears on the shape, as shown in Fig. 24. The box initially displays the current text as printed in the shape. An additional text edit box is also created on the right of the shape if it already contains an accompanying comment box 435. The user can then proceed to edit the text description inside the shape.

To edit the text within the rectangular comment box on the right, the user simply presses a key, for example the 'Tab' key. The 'Tab' key is used to toggle the text edit cursor between the shape and comment edit boxes. To terminate the in-place edit operation, the user can either press another key, for example the 'Return' key, or click anywhere in the background of the flow chart editing window 305 with the help of a mouse.

This Sheet Has Been Replaced



When the target label 422 within a branch instruction is modified to point to another label 425, the branch shape's 421 outgoing branch line 423 would automatically be redirected towards to the new label shape 425. This is handled by the text in-place edit 405 and the branch line removal 404 and creation 403 sub-systems.

When the text in-place edit sub-system 405 is invoked, an instruction text buffer and a comment text buffer is created. The instruction and comment text data are derived from the shapes.

User input from the keyboard is channelled into the buffers. Whenever a key is pressed, the sub-system 405 examines the new state of the buffer text. The presence of the 'Tab' character is checked so that the text edit cursor can be redirected to either the shape or comment edit boxes. In addition, the text buffer in which the cursor originated from has to be modified to remove the 'Tab' character.

If the 'Tab' character is not present within both the instruction and comment buffers, a search is done to check for the presence of the 'Return' character. A 'Return' character denotes the completion of the text in-place editing operation. If found, the 'Return' character is removed from the appropriate buffer. All the leading and trailing spaces in both buffers are trimmed. The buffer text is then written back to the text string members of the shape.

The contents of the instruction buffer are then scanned for the presence of branch instructions with the help of the assembly language database sub-system 202. If a branch instruction is detected, the current outgoing branch lines 423 of the shape 421 will be removed and new branch lines 427 will be added. This process is needed in order to handle the situation whereby a branch instruction's target label is changed.

Finally, the temporary buffers and text edit boxes are deleted from memory and the flow chart editing window 305 is refreshed.

Branch lines 423 are automatically removed and re-created when text in-place editing is carried out on a branch shape 421. With this feature, the user no longer needs to manually redirect the branch line 423 when the target label changes. The sub-systems involved include the branch line removal 404 and creation 403 sub-systems. The branch line removal sub-system 404 identifies the branch lines 423 whose tail are connected to the branch shape 421 and removes them. On the other

This Sheet Has Been Replaced



This Sheet Has Been Replaced



hand, the branch line creation sub-system 403 examines the branch instruction for its target label definition and subsequently produces a new branch line 427 linking the branch shape 426 to its new label shape 425.

Fig. 25, 26 and 27 demonstrate the states of an arbitrary branch shape before, during and after a text in-place edit operation involving a label change. Prior to the operation, the branch shape is connected, via a branch line, to a DEF label shape 422 as shown in Fig. 25. The target label is then changed to ABC 425 as shown in Fig. 26. When the 'Return' key is pressed, the sub-system 404 proceeds to delete the old branch line 423. A new branch line 427 is then created, which points to the new label shape 425, as shown in Fig. 27.

The branch line removal sub-system 404 is activated each time the text in-place editing sub-system 405 encounters a branch instruction. The flow chart depicting the operation of the sub-system 404 is shown in Fig. 20. The branch shape's 421 size is initially inflated by a pre-determined amount. The sub-system 404 then proceeds to inflate the sizes of all the branch lines that occur at the same level as the shape 421. As each relevant branch line is inflated, a test is done to ascertain whether its enlarged area overlaps with the shape 421. The branch line 423 is deleted if it overlaps. Fig. 15 presents a diagram illustrating the situation whereby an overlap in area is detected between a shape 321 and a line 322.

The branch line creation sub-system 403 is also invoked each time the text in-place editing sub-system 405 identifies a branch instruction. It is activated after the branch line removal operation.

A typical assembly language branch instruction comprises the branch instruction name 441, followed by the target label location 442, as shown in Fig. 28. In an unconditional branch, the program flow is simply directed to the instructions beginning at the specified label. In the case of a conditional branch, the program flow either progresses to the next instruction or to the instructions beginning at the specified label 442, depending on some run-time conditions.

The branch line creation sub-system 403 extracts the target label name 442 from the branch instruction. The entire flow chart database 10 is searched for the presence and the location of the label shape 442. When the matching label shape 442

This Sheet Has Been Replaced

is found, a calculation is made to determine the distance between the branch and label shapes. This value is used to construct the branch line 427. The degree of protrusion of the branch line from the right edge of the flow chart shape column is modified when the sub-system finds that it overlaps with an existing branch line. In addition, the parent index array contents of the branch shape are copied to the branch line's parent index array member. The schematic flow chart of the operation of the branch line creation sub-system 403 is shown in Fig. 21. If no matching label shape 442 is found in the database 10, a warning message indicating a non-existent label is issued to the user. All labels within the flow chart database 10 are unique. If a duplicate label is encountered in the code during the conversion of assembly codes to its flow chart, an error message is produced. In addition, if the user edits a label shape's text such that it duplicates another existing label within the database 10, an error message is likewise produced.

There are two ways of inserting additional flow chart shapes and lines into an existing flow chart. They include the manual drawing method and the automated insertion method.

In the manual drawing method, new flow chart objects are drawn by first selecting the appropriate flow chart shape and line icons from the system's toolbar. The mouse is then used to specify the location and size of the object. Thereafter, text is entered into the shape by using the text in-place editing function. New flow chart layers can also be added. There are eight types of flow chart shape icons namely, label, non-branching, unconditional branching, conditional branching, call, return macro and block-text icons. There are two types of line icons namely, connecting line and branch line icons. These icons provide the means to fully represent the assembly codes in the form of flow charts. The manual method provides the greatest flexibility to the user.

The automated insertion of shapes can be accomplished in one of two ways. They are the insertion by text and insertion by shapes methods.

The insertion by text method is the most convenient method available for the addition of flow chart objects into an existing flow chart. The user begins by entering assembly codes into a text input box. The text is subsequently converted into an

This Sheet Has Been Replaced

equivalent flow chart. This flow chart segment is then inserted into the existing flow chart at the desired location. New flow chart levels can also be easily created. Fig. 29 shows an overview of the insertion technique.

The flow chart insertion by text operation in the flow chart insertion sub-system 401 is handled by the text input sub-system 451, an instruction scanning and storage sub-system 452, a function detection sub-system 453, a vertical offset calculation sub-system 454, a connecting and branch line creation sub-system 455, a parent index assignment sub-system 456, a branch line across levels detection sub-system 457, an index assignment sub-system 458, a line number generation sub-system 459, a branch target label line number generation sub-system 460 and a functions check sub-system 461. These sub-systems operate one after another in the sequence stated above. The flow chart illustrating the top-down interaction between the sub-systems is shown in Fig. 29.

Initially, the user has to select the point of insertion, which must be a vertical connecting arrow that can be from any level in the flow chart. The right mouse button is then pressed to produce a floating context menu. When the relevant menu item to initiate the insertion process is chosen from the context menu, a dialog box will be shown on the screen. This dialog box contains a text input box that allows the user to enter text. All the common text editing operations are supported. It is also possible to copy or cut a section of code from another text editor and paste it into the box. A button is available for the user to clear the entire contents of the text box. Labels, new functions, branch instructions and other assembly program constructs can be added. A vertical scroll bar will automatically be produced on the right of the text box when the text exceeds the height of the text input box. A button within the dialog box is available that allows the user to abort the insertion operation. The user is prompted for confirmation to abort if text is present within the text box.

When the necessary text has been added, the user closes the dialog box. This is followed by a series of progress status dialog boxes, which show the status of the various operations that are needed to insert the text into the flow chart. Any error encountered is reported to the user. At the end of the series of operations, the flow chart editing window 305 is refreshed to show a new flow chart with the newly

This Sheet Has Been Replaced



inserted flow chart shapes and lines. New branch lines and flow chart layers are automatically added where necessary.

The text input sub-system 451 is responsible for extracting and subsequently manipulating the text by converting it into a form suitable for the subsequent insertion operations. Text manipulation involves the removal of unnecessary tabs and spaces, and the insertion of line breaks between the lines. This is necessary because the text retrieved from the box is a continuous string. Line breaks help to ensure the correct identification of every individual line.

The flow chart depicting the operation of the instruction scanning and storage sub-system 452 is shown in Fig. 30. This sub-system 452 is responsible for converting the lines of text that were previously retrieved from the dialog box into their equivalent flow chart shapes and records. It scans the text to search for instruction, label, macro and comment content to create shapes for the flow chart editing window 305 as well as draw objects for the flow chart database 10.

Every line of code is examined sequentially. Empty lines are ignored. A line of text consisting of only comments without any instructions is designated as a block-text shape. Consecutive lines of a similar nature are appended to the previously created block-text shape. Therefore, a block of comments is represented by one large block-text flow chart shape rather than a series of small comment shapes. The comment text will be displayed alongside the instruction's shape for text lines with instructions on the left and accompanying comments on the right. The comment text will follow the instruction's shape wherever it is placed.

The text line is next examined for label and macro name content. If any of these names are present, the appropriate label or macro shape objects are created in the flow chart editing window 305 and database 10 respectively. Macro names are stored in the macro database. The information within the macro database is derived by scanning the header files that were defined in the original assembly code files. The header files contain macro definitions. A search is conducted in the macro database when the text scanner encounters an unknown name. The name is treated as a label when a macro name match is not found. The label records are stored in a manner that allows them to be easily matched with the target labels of branch instructions. This

This Sheet Has Been Replaced

facilitates the construction of branch lines in the connecting and branch line creation sub-system 455.

Finally, the text is inspected for instruction content. Depending on the instruction type found, the corresponding shape and object are produced. The instruction types represent the major categories of assembly instructions. The selection of instruction types is comprehensive enough to fully represent any set of assembly codes in the flow chart domain. A vertical connecting line is always created after a flow chart shape has been added. However, no line is created for return shapes, as the program flow does not proceed further from it. When this operation ends, the next line is fetched for processing. The function terminates when all the lines of code have been converted to flow chart shapes and objects.

The function detection sub-system 453, whose flow chart is shown in Fig. 31, derives its input from the flow chart database 10. This function is primarily responsible for creating the hierarchical flow chart structure.

The sub-system 453 begins by accessing the database 10 to look for shapes belonging to the call category. All other types of shapes are ignored. The short-listed call shapes' parent index array are then examined. Only the call shapes that do not contain any parents in their parent index array are processed. The rest are disregarded. This is because the newly inserted shapes do not have any parents assigned to them yet, unlike the existing shapes. This feature is used to differentiate the new shapes from the existing ones.

Each call shape's target label is extracted. If the label is absent from the database 10, the user is notified of the non-existent label. If the label is present, its corresponding object is retrieved from the database 10 and its unique index is noted. The shape co-ordinates contained in the label object are also modified such that the label shape will be situated at the top of the flow chart window. The label shape's index is then incremented to obtain the following object. That is, a search is conducted to identify the object whose index is larger than the label shape's index by one. If the next shape is not of the return type, its co-ordinates are modified so that it falls below the preceding label shape to form a progressively expanding flow chart. This process continues until a return object is found. It is assumed that function

This Sheet Has Been Replaced

definitions begin with a label and end with a return. By now, a flow chart representing the call function has been constructed. The last step involves the addition of the call shape's index to the parent index array of the function shapes. In this way, an association is created between the call shape and its function flow chart. To view the call function's flow chart, the call shape's index is extracted and used to refresh the flow chart window to display only those shapes and lines with the call shape's index in their parent index arrays.

The function detection sub-system 453 ends when the function flow charts for all the new call shapes in the database 10 have been created.

When a connected group of flow chart shapes is added into an existing flow chart, the shapes that are situated below the point of insertion have to be shifted down by an amount equal to the height of the inserted flow chart. Consequently, all connecting and branch lines have to be readjusted accordingly. The flow chart of this operation in the vertical offset calculation sub-system 454 is shown in Fig. 32.

The first step involves the calculation of the vertical height of the inserted flow_chart shapes and lines for each affected level. All the objects from the database 10 are accessed one at a time to find those objects whose parent index array are empty. These are the newly inserted objects. The co-ordinates of every newly inserted shape or line found are noted. If a certain shape's co-ordinates exceeds the previous known boundary of the group of inserted shapes for a particular level, the boundary is enlarged. The boundary grows steadily as the co-ordinates of the inserted shapes are progressively checked. Therefore, the maximum height occupied by the inserted shapes is known only after the whole database 10 has been accessed. There is no necessity to determine the width of the inserted shapes as the flow chart is assumed to appear as one column.

The next operation involves finding the existing objects within the database that belong to the current display level of the flow chart editing window 305. Shapes and lines that do not belong to the current display level of the window 305 are not displayed. That is, if the display level is set to 10, only those shapes with a member of value 10 in their parent index array will be displayed. The co-ordinates of the existing shapes and lines belonging to the current display level are checked to ascertain

This Sheet Has Been Replaced

whether they are situated above or below the point of insertion. Shapes and lines above the point are ignored. Shapes and lines that appear below the insertion point are shifted downwards by an amount equal to the height the inserted flow chart section. Existing branch lines are adjusted accordingly when the inserted shapes fall between their source and destination points.

The function ends when all the affected flow chart shapes and lines in all the levels have been shifted downwards.

The connecting and branch line creation sub-system 455 creates connecting lines as well as branch lines, which connect a branch shape to its target label shape. Different types of conditional and unconditional branch instructions are present in assembly languages. Each uses a different number of arguments. In addition, the label prefix varies according to whether it is a long or a short branch. Therefore, the text scanner must be able to handle the many varieties of branch instruction syntax for different assembly languages, and correctly extract the target label's name. The algorithm for the creation of the branch lines is shown in Fig. 33.

A conditional branch shape has one connecting line pointing towards it and two other lines emerging from it. One of the emerging connecting lines simply points to the next shape below, while the other is a branch line that points to the branch instruction's target label shape. An unconditional branch has only one branch line emerging from it.

The flow chart shape database 10 is searched for all instances of conditional and unconditional branch objects. When a branch shape is encountered in the search, a test is done to determine whether it already has a branch line emerging from it. This is accomplished by first expanding the area occupied by the branch shape. After which, the co-ordinates of all the existing branch lines belonging to the same level are checked to determine whether any overlap exists between the branch shape and the branch lines. Branch shapes with branch lines already assigned to them are ignored. The function then continues to check the next branch shape. The function ends when all the branch shapes have been examined.

The target label is obtained for those branch shapes that do not have any emerging branch line. This target label is used as a search field to retrieve the

This Sheet Has Been Replaced



corresponding label object from the database. The user is notified if the label object is not present in the database. When the label object is found, the co-ordinates of both the branch and the label shapes are compared to determine whether an upward or downward branch line has to be created. The protrusion of the branch line away from the right edge of the flow chart shape column has to be calculated such that it does not overlap the path of any existing branch line. A reasonable distance is placed between two vertically adjacent branch lines. The overlapping of branch lines may result when there are a large of number of branch lines occupying the same area. To overcome this problem, the colour of the branch lines are changed when selected by the user, so that the desired branch line can be easily identified.

The height, width and direction of the desired branch line thus computed are used to construct the branch line. The branch line's arrow head is automatically orientated to point towards the label shape. Finally, the entire parent index array of the branch shape is copied onto the parent index array of the branch line. This is necessary because the branch line has to be displayed together with both the branch and label shapes. Branch lines that begin and end at different levels in the flow chart are addressed by the branch lines across levels detection sub-system 457.

The parent index array of newly inserted shapes is empty. The parent index assignment sub-system 456, whose flow chart is shown in Fig. 34, is responsible for adding parent indexes to those arrays. The parent indexes to be added are obtained by copying the array of parent indexes of any shape or line that are already present in the current display level of the flow chart editing window 305. This algorithm works well regardless of whether the shapes were added at the root level or at the lower levels of the flow chart. The display level number of the root level is 0. This number increases for the lower levels. The parent index assignment sub-system 456 is called after the discrimination of new shapes from the existing ones by way of an empty parent index array is no longer required.

The database 10 is scanned for the first object whose parent index array contains the current display level number. Scanning stops after such an object has been found. The contents of the parent index array of this particular object is stored in a temporary array. The database is then scrutinised to look for objects with an empty

This Sheet Has Been Replaced

parent index array. This includes both shapes and lines. When these objects are found, the contents of the temporary array obtained earlier are copied to their parent index arrays.

The function ends when the whole database 10 has been inspected. The result is that all objects will have at least one parent.

Assembly language programming permits branches from instructions within a function to another instruction located in other functions. No structured programming style can be enforced by virtue of the nature of the language. This leads to the problem of wayward branch lines that emerge from a branch shape in one flow chart level and terminate at a label shape in another level. Such branch lines must be detected to enable the system to take the necessary course of action. This is handled by the branch line across levels detection sub-system 457.

The first portion of the sub-system 457 operation essentially compares the parent index arrays of both the branch shape and its target label shape. The method employed in getting the branch and label objects is similar to that used in the connecting and branch line creation sub-system 455. If the array contents are similar, the branch line concerned would therefore begin and terminate at the same flow chart level.

The remaining portion of the sub-system deals with the situation whereby both the parent index array contents of the branch line and the label shape do not tally with each other. The means that the branch line starts and ends at different levels. When this occurs, the branch line is deleted and the user is notified. The branch line concerned is found by scouting for all the branch line objects in the database 10 that have the same parent index array as the branch shape. These branch line objects are short-listed. The area occupied by the branch shape is then expanded. The short-listed branch lines are then tested to ascertain whether any of them intersect with the expanded branch shape. If an intersection is found, the branch line concerned is deleted. The function then proceeds to look for the next branch line that traverses across levels. The flow chart of this operation is shown in Fig. 35.

The purpose of the index assignment sub-system 458 is to detect the connectivity and program flow direction amongst the shapes and lines. Each flow

This Sheet Has Been Replaced

chart object has a previous index and a next index member. The previous index member stores the index of the shape or line preceding it, while the next index stores the index of the shape or line following it. The first shape within a flow chart level will have a previous index of 0. Likewise, the last shape will have a next index of 0. This is true for all the flow chart levels. This function tackles one flow chart level at a time, until all the levels have been examined. The previous and next indexes are used for assigning line numbers to shapes in the subsequent line number generation sub-system 459 and also for generating text codes from the flow chart by the flow chart to assembly code conversion engine sub-system 300.

The index assignment sub-system 458 begins its operation by initialising all the flow chart objects' previous and next indexes to 0. This erases the effects of any previous execution of this sub-system 458. When the flow chart is edited by adding or deleting both shapes and levels, the formerly assigned previous and next indexes in the objects will no longer be valid. The largest index of all the objects is derived from the database 10. This index represents the maximum number of possible levels in the flow chart. The total number of objects is next obtained. The numerical values of the largest index and the total number of objects might not tally when objects have been added or deleted from the flow chart. This is because the index number that is given to newly created objects is ever increasing. Index numbers that are lost due to deleted objects are not recycled.

Two nested iterations are used. The first iteration ensures that all the possible flow chart levels are examined. The second iteration, which is nested within the first, guarantees that the entire database 10 is searched to locate all the objects that exist at a particular level. The following explanation describes the algorithm, whose flow chart is shown in Fig. 36, for assigning indexes to shapes and lines in an arbitrary level of the flow chart.

The second iteration mentioned above cycles through all the objects in the database 10. Each successive object's parent index array is examined to ascertain whether it belongs to the current display level. After which, the parent index arrays of the objects belonging to the level concerned are further scrutinised to establish whether the same objects have been previously accessed by their parent function at

This Sheet Has Been Replaced



another level. This step is necessary to prevent an object's previous and next indexes from being assigned values twice. Suppose the objects at a particular level have parents whose numbers are 1 and 2. When the algorithm reaches level 2, it has already assigned the previous and next indexes for objects in level 1. It is therefore not necessary to repeat the operation again for objects in level 2, as all the objects which contain a parent index of number 2 already have a parent index array member with number 1. The series of sub-systems prior to this index assignment sub-system 458 ensures that this condition is always true. Before the sub-system 458 proceeds further, the parent index array of the object is checked to see if there are members whose number is smaller than that of the current level being examined. If there are such members, the current level is skipped.

If an object passes the test above by not having any smaller members in its parent index array, the sub-system 458 proceeds to the next stage. Every object at a particular level is successively considered as a reference object. Its co-ordinates are expanded by a pre-determined amount. Following this, the co-ordinates of the rest of the objects in the same level are expanded one at a time. A test is done to check for any intersection between the reference object and any of the other objects at that level. If an intersection is detected, a check is further done to see whether it occurs between a line and a shape, and not between a line and a line, or a shape and a shape. Any anomalies are highlighted to the user. The connecting line's arrow location and direction are noted when assigning the previous and next index members of both the line and the shape.

The line number generation sub-system 459 is responsible for assigning consecutive line numbers to every flow chart shape. The flow chart of the operation of the sub-system 459 is shown in Fig. 37. The line numbers appear at the space to the left of every shape. They follow the shape wherever it is placed. These numbers are especially useful for tracing program flow involving branch shapes. This is because branch shapes have their target label's line number appended to their text. This is helpful in locating the branch's label in the printout.

The method of accessing the objects at all the possible levels in the flow chart is similar to that used by the index assignment sub-system 458. To monitor the

This Sheet Has Been Replaced

assignment of line numbers, a counter, which starts from 1, is used. It is incremented by 1 after its value is assigned to a shape. Every shape within all the levels of the flow chart will have a unique line number. For each level, the shape with a previous index equal to 0 is sought. When it is found, it is assigned a line number by the counter. The value within the object's next index member is then used to get the following object and so forth. This process continues until an object with a next index equal to 0 is found. This is the last object at that particular level. The sub-system then proceeds to the next level, until all the levels have been accessed. Line numbers are not assigned to connecting and branch lines because they do not constitute an instruction in the code. However, index numbers are given to all objects, which include both shapes and lines.

The line number generation sub-system 459 is applied to the database 10 prior to any flow chart printing operation. This is to enable the printout's shapes to have updated line numbers. The line numbers are omitted from the text output generated by the flow chart to codes conversion operation.

To facilitate the debugging and tracing of program flow, text within all the branch shapes have their target label's line number 443 appended to them with the help of the branch target line number generation sub-system 460. This allows the user to quickly locate a branch shape's target label shape by searching for the label with the matching line number, rather than tracing the destination of the branch line. This is useful for inspecting the flow chart on the screen and from the printout. The appended line number 443 is omitted when the flow chart is converted into text codes. This is necessary to prevent the assembler from issuing syntax error messages for branch instructions. Fig. 38 shows the algorithm used by the sub-system 460 in the form of a flow chart.

This sub-system 460 extracts all instances of conditional and unconditional branch objects from the database 10. The target label within each branch shape's instruction text is noted. The database 10 is subsequently searched to obtain the label's object, using the label name as a search field. As line numbers have been previously assigned to every shape by the line number generation sub-system 459, the label's line number is easily obtained after getting its object. This particular line number 443 is

This Sheet Has Been Replaced



then appended to the instruction text of the branch shape. If a label is not found, a number 0 is given to the branch shape and the user is notified of the missing label.

The purpose of the functions check sub-system 461 is to inspect the flow chart for the existence of both single level and multiple level recursive calls after the new flow chart segment has been added. Fig. 39 illustrates the operation of this sub-system 461. In the system's navigator tree 106, a tree node 108 representing a call shape can be further expanded to produce a branch 107. This branch 107 contains nodes representing all the shapes that make up the function. In a single level recursive call, this expanded branch will in turn contain a node that calls the very same function in which it is already located. This condition constitutes a single level recursive call. Therefore, if the user were to continue examining the details of the call node, an infinite number of branches will be produced. Detecting recursion in the flow chart would enable the system to suppress the expansion of nodes that produce infinitely large branches. The same problem occurs for recursion across multiple levels.

The sub-system 461 begins its operation by searching for the presence of any single layer recursion. The database 10 is scanned for all instances of call objects. The index of each call object found is noted. The value of the members within the object's parent index array is then examined to see if any of them are similar to object's own index. If a match is detected, single layer recursion has occurred. This is because the call object is both its own child and parent. The user is notified and the call object's ability to further expand its branch in the navigator tree 106 is disabled. The sub-system 461 continues by examining the next call object.

If single layer recursion is not observed, the sub-system 461 proceeds to look for any incidence of multiple layer recursion. This is done by first obtaining the call's target label object. The parent index arrays of both the call and label object are compared. If they are found to be different, a search is conducted on the call object's upper layers to look for duplicate calls with similar target labels. As the number of upper layers is not known, a recursive function is used to intelligently traverse towards the root level of the flow chart and backtrack properly to the original call object. If an object with a similar call label is found along the way to the root level, multiple layer recursion has occurred, and the user is notified.

This Sheet Has Been Replaced



In addition to adding flow chart segments by way of text, the user can also copy or cut a section of a flow chart and insert it into another location within the same flow chart, or onto another flow chart in a separate flow chart editing window 305. Therefore, entire flow chart structures or just small segments of a flow chart can be easily copied and transferred to other charts. The copying and subsequent insertion of flow chart segments across layers in a flow chart is also permissible.

The copying and pasting of flow chart segments 471 involves the exchange of flow chart object linking and embedding (OLE) data between the system's database 10 and the operating system's clipboard memory. The size of this memory is determined by the available resources in the computer. When a new section of the flow chart is copied, the new data overrides the previous data that was held in the clipboard memory. However, the same clipboard data, if not overridden, can be pasted or inserted any number of times to various locations in any existing flow chart.

To insert a segment of flow chart onto a particular location, it must first be copied into memory. This is done by using the mouse to define a rectangular region that encompasses the desired flow chart segment. A particular shape or line does not need to be fully enclosed by the rectangle for it to be selected. After which, all the flow chart shapes and lines within that rectangle are automatically selected. The user can then copy the selected flow chart segment into the clipboard memory.

The point of insertion must always be a vertical connecting line. When the line is selected, the user presses the right mouse button to activate a floating context menu. The relevant menu item to initiate the insertion process is then chosen. This is followed by a series of progress dialog boxes that inform the user of the sequence of operations that are occurring. The flow chart in Fig. 40 shows an overview of the insertion by shapes algorithm.

The flow chart insertion by shapes operation in the flow chart insertion sub-system 401 is handled by the index assignment for copied shapes sub-system 472, flow chart to assembly codes conversion for copied shapes sub-system 473 and the flow chart insertion by text sub-system 474.

The algorithm adopted to assign indexes to the inserted shapes and lines is almost similar to that used by the index assignment sub-system 458 mentioned earlier.

This Sheet Has Been Replaced

In that particular sub-system 458, all the objects' previous and next indexes were allocated values basing upon their connectivity and flow direction with respect to each other. In the assignment of indexes for copied shapes sub-system 472, only the objects from the newly inserted flow chart shapes are addressed, and not the entire database 10. The first shape will have a previous index of 0, while the last shape will have a next index of 0. The function ends when all the inserted shapes and lines have been assigned indexes.

The assembly code to flow chart conversion for copied shapes sub-system 473 is responsible for converting the inserted shapes into text. Only the inserted shapes are processed, and not the entire database 10. Using the inserted objects' previous and next indexes derived from the index assignment operation, a sequence of text instructions is pieced together that represents the text equivalent of the inserted flow chart. The format of the output text generated by this function is similar to the format of the text that is extracted from the text input dialog box by the text input sub-system 451 mentioned earlier. The text is then channeled to the flow chart insertion by text sub-system 474.

This Sheet Has Been Replaced



Claims

1. A flow chart engine that is used in a graphical microcontroller software development system, said engine comprising:

a flow chart navigator engine sub-system;

an assembly code to flow chart conversion engine sub-system;

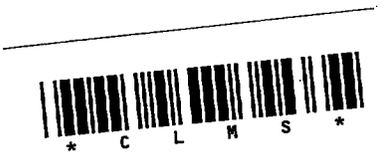
a flow chart to assembly code conversion engine sub-system; and

a flow chart editing engine sub-system.

2. A sub-system according claim 1, wherein the said flow chart navigator engine sub-system comprises:

a tree branch expander-contractor sub-system;

a tree parent node state setter sub-system;



This Sheet Has Been Replaced

a tree branch adder sub-system;

a tree refresher sub-system; and

a tree node selection handler sub-system.

3. A sub-system according to claim 2, wherein the said tree branch expander-contractor sub-system manages the expansion and contraction of branches in the navigator tree structure to enable users to selectively reveal or hide additional flow chart details in the lower flow chart layers.

4. A sub-system according to claim 2, wherein the said tree parent node state setter sub-system tracks the status of parent nodes which may either be in the expanded or contracted state by modifying certain attributes of the corresponding flow chart parent shapes.

5. A sub-system according to claim 2, wherein the said tree branch adder sub-system inserts flow chart nodes that belong to a certain parent node and to sort the tree nodes according to the relative positions of their corresponding shapes in the flow chart.

6. A sub-system according to claim 2, wherein the said tree refresher sub-system automatically updates the hierarchical tree structure whenever a change in the corresponding flow chart occurs, such that the latest tree state information is always available to users, with the help of the tree branch expander-contractor sub-system in claim 3, the tree parent node state setter sub-system in claim 4 and the tree branch adder sub-system in claim 5.

7. A sub-system according to claim 2, wherein the said tree node selection handler sub-system is activated whenever the user selects a particular node from the navigator tree, causing a message to be sent to the flow chart editing window

* C L M S *

This Sheet Has Been Replaced

to display only the flow chart objects at the chosen level and to also highlight the selected flow chart shape.

8. A sub-system according to claim 1, wherein the said assembly code to flow chart conversion engine sub-system comprises:

a text extractor sub-system;

an assembly instruction database sub-system;

an instruction scanning and storage sub-system;

a function detection sub-system; and

a connecting and branch line creation sub-system.

9. A sub-system according to claim 8, wherein the said text extractor sub-system identifies the windows containing the codes that are chosen for conversion into flow charts, retrieves and stores the pointers to these windows, in order for the system to access the code text, and to also manipulate the text strings into a form suitable for instruction scanning.

10. A sub-system according to claim 8, wherein the said assembly instruction database sub-system permits the user to select the assembly language of choice and subsequently loads the appropriate assembly language instruction set into the system and stores each instruction into a form that allows extracted instructions from the code window to be easily identified and categorised.

11. A sub-system according to claim 8, wherein the said instruction scanning and storage sub-system intelligently distinguishes between labels, instructions and comments with the help of the assembly instruction database sub-system in claim 10, segregates the instructions into categories and finally stores them.

* C L M S *

This Sheet Has Been Replaced

12. A sub-system according to claim 8, wherein the said function detection sub-system analyses the code for instances of functions and sub-functions, in order to assign different segments of the flow chart to different levels, resulting in a hierarchical flow chart structure.

13. A sub-system according to claim 8, wherein the said connecting and branch line creation sub-system generates connecting lines between successive shapes to denote program flow, automatic matching of branch shapes with their respective target labels shapes so as to provide co-ordinates for the branch line to be created and methods to reassign the co-ordinates of branch lines to prevent them from overlapping each other.

14. A sub-system according to claim 1, wherein the said flow chart to assembly code conversion engine sub-system comprises :

a connectivity detection sub-system;

- a flow chart traversing sub-system; and

a text output sub-system.

15. A sub-system according to claim 14, wherein the said connectivity detection sub-system checks for the merging of shapes and lines, assigns the relevant indexes to the shapes and lines to indicate the sequence of program flow, and to also warn the user of unconnected shapes.

16. A sub-system according to claim 14 wherein the said flow chart traversing sub-system is able to intelligently track the course of program flow in a logical manner within the hierarchical flow chart, ensuring that all levels are examined.

17. A sub-system according to claim 14 wherein the said text output sub-system identifies the correct code window to channel the text to, extracts the text from

* C L M S *

This Sheet Has Been Replaced

the shapes, whose sequence is determined by the flow chart traversing sub-system in claim 16, and to automatically indent the text output to increase the legibility of the code.

18. A sub-system according to claim 1, wherein the said flow chart editing engine sub-system comprises:

a flow chart drawing sub-system;

a flow chart insertion sub-system;

a text in-place edit sub-system;

a branch line removal sub-system; and

a branch line creation sub-system.

19. A sub-system according to claim 18, wherein the said flow chart drawing sub-system is responsible for drawing flow chart shapes whose sizes and locations can be easily changed by the user and also adjusted dynamically by the sub-system to prevent the shape edges from falling in between grid squares, and for drawing straight and right-angled connecting lines whose sizes, location and orientation can be easily modified by the user.

20. A sub-system according to claim 18, wherein the said flow chart insertion sub-system comprises:

a text input sub-system;

an instruction scanning and storage sub-system;

a function detection sub-system;

a vertical offset calculation sub-system;

a connecting and branch line creation sub-system;

a parent index assignment sub-system;

a branch line across levels detection sub-system;

an index assignment sub-system;

* C L M S *

a line number generation sub-system;

a branch target label line number generation sub-system; and

a functions check sub-system.

21. A sub-system according to claim 20, wherein the said text input sub-system provides a text edit window that enables users to enter codes by either typing directly or by transferring codes from other sources through copying and pasting operations, and to also manipulate the codes into a form that is suitable for the instruction scanning and storage sub-system in claim 22.

22. A sub-system according to claim 20, wherein the said instruction scanning and storage sub-system intelligently differentiates between labels, instructions and comments by making references to the assembly instruction data sub-system in claim 10, categorises the instructions and stores them in a temporary buffer that is independent of the main database that holds all the existing flow chart objects.

23. A sub-system according to claim 20, wherein the said function detection sub-system examines the flow chart objects in the temporary buffer storage for instances of functions and sub-functions, to create new flow chart levels accordingly, and to also ensure that these new levels will merge correctly with the existing flow chart.

24. A sub-system according to claim 20, wherein the said vertical offset calculation sub-system computes the dimensions of the flow chart objects in the temporary buffer for all the newly created levels, shifts the objects in the existing flow chart by an amount equal to the area to be occupied by the inserted objects, and also readjusts all existing connecting and branch lines.

25. A sub-system according to claim 20, wherein the said connecting and branch line creation sub-system generates connecting lines between successive shapes

This Sheet Has Been Replaced

in the temporary buffer, branch lines between the branch shapes in the temporary buffer to any target label shape in the buffer or the existing flow chart, and to also manage the branch lines to prevent them from overlapping each other.

26. A sub-system according to claim 20, wherein the said parent index assignment sub-system designates parent indexes to the flow chart objects in the temporary buffer based upon the level in the hierarchical flow chart where they will be inserted, and to also maintain the integrity of the flow chart structure by copying the parent indexes of the existing objects, that are situated in the same level, into the parent index array of the objects in the buffer.

27. A sub-system according to claim 20, wherein the said branch line across levels detection sub-system locates branch lines that begin from a branch shape in one flow chart level and terminate at a label shape that is situated at another level, so as to enable the system to issue warning messages to the user.

28. A sub-system according to claim 20, wherein the said index assignment sub-system merges the flow chart objects in the temporary buffer with those in the existing flow chart for all the levels, checks for connectivity between all the shapes and lines and assign the relevant indexes which are used to indicate the sequence of program flow and to also warn the user of unconnected shapes.

29. A sub-system according to claim 20, wherein the said line number generation sub-system is responsible for assigning consecutive line numbers to every flow chart shape in all the flow chart levels, to help the user in tracing program flow, especially when branch shapes are involved.

30. A sub-system according to claim 20, wherein the said branch target label line number generation sub-system extracts the target label in the branch instructions of all the conditional and unconditional branch shapes in the flow chart,

* C L M S *

obtains the line numbers of the label shapes, and appends the numbers to the corresponding branch instruction text, to allow the user to quickly locate the target label of a branch shape.

31. A sub-system according to claim 20, wherein the said functions check sub-system serves to inspect the flow chart for the existence of both single level and multiple level recursive calls after the new flow chart segment has been inserted, in order to prevent the situation in which an infinite number of navigator tree branches would be produced if the user were to continue examining the details of a tree node that is recursive in nature.

32. A sub-system according to claim 18, wherein the said text in-place edit sub-system allows users to edit the text contained within flow chart shapes and their associated comment boxes, by creating editable text boxes that are placed over the shape and to also automatically redirect the branch line from a branch shape if the target label for the branch instruction is changed, with the help of the branch line removal and creation sub-systems in claims 33 and 34 respectively.

33. A sub-system according to claim 18, wherein the said branch line removal sub-system deletes outgoing branch lines from the branch shape concerned by identifying the branch lines that are connected to the branch shape from among all the branch lines that exist in the flow chart level where the branch shape is located.

34. A sub-system according to claim 18, wherein the said branch line creation sub-system generates branch lines between branch shapes and their target label shapes by extracting the target label definition from the branch instruction and locating the corresponding label shape to calculate the dimensions and location of the branch line.