9800772-7

Graphical Microcontroller Software Development System

REPLACED SHEETS SG
☑ Description : paged 1 to paged 18
☑ Claim(s) : paged 19 to paged 24
☐ Drawing(s) : paged ..... to paged .....
☑ Abstract : paged - to paged 25

Field of the Invention

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

Description of the Prior Art

Assembly language code development for microcontrollers has traditionally been text based. The programmer edits the code using a conventional text editor. After which, the codes are compiled by the assembler. The programmer makes the necessary corrections to the code, based on the warning or errors messages issued by the assembler. The codes are then loaded onto the in-circuit-emulator where run-time errors are noted and corrected by modifying the codes. This is an iterative process.

Most conventional text editors used display text in one colour, so that it is difficult to visualise the structure of the assembly codes. Programmers overcome this inherent problem by liberally using comments and blank lines throughout the code to segregate and identify different logical sections.

Assembly language programming involves the use of mnemonics. These are often cryptic, as the function of the instruction is abbreviated to form the mnemonic name. Programmers have to insert substantial comments to make the code more understandable.

Very often, the overall program structure and flow is known only to the original programmer. A programmer not involved in the coding will require much effort and time to understand them, despite the use of comments. The nature of

This Sheet Has Been Replaced



assembly language makes the debugging and tracing of algorithms or program flow difficult and time consuming.

Many microcontrollers targeted for consumer products have a very limited built-in memory size to lower its cost. In an effort to fully utilise the available memory and to pack as many software features as possible into the product, programmers have to resort to using unconventional, unstructured programming practices to reuse sections of code, or to employ certain program organisations that use less memory. This results in convoluted code that is difficult to comprehend and hence troubleshoot. Such programming practises are discouraged, but are still adopted because of memory size constraints.

Existing computer-aided software engineering (CASE) tools are capable of generating flow charts from assembly language codes. However, the flow charts created are static, and cannot be edited. They merely provide a graphical representation of the program. The screen layout of an existing non-editable, single layer flow chart generator system 200 is shown in Fig. 1. The generated flow chart consists of flow chart objects which are directly mapped from the text instructions. There are five types of flow chart shapes namely, call 201, label 202, decision 203, non-decision 204 and return 205. There are also accompanying comments 206, branch lines 207, connecting lines 208 and line numbers 209. The flow chart is made up of only one layer. To access the flow chart objects representing instructions within a function, the flow chart has to be manually scrolled to the appropriate position elsewhere on the same layer. This is inefficient for large programs with lengthy flow charts. The flow chart in Fig. 1 contains a call 201 to a function whose instructions begin at another location denoted by label 202. No flow chart navigational tools are available to automatically scroll the flow chart to the desired location. Colour syntax highlighting for assembly languages is also absent in text editors. No editing in the flow chart domain, nor subsequent conversion back to the source codes is possible. The disadvantage of such tools is that when program errors are found in the flow chart, the necessary correction can only be made in the assembly code text.

Software tools are currently available to generate assembly language codes from flow-based diagrams, using proprietary graphical notation, rather than

This Sheet Has Been Replaced



established flow chart shapes and structures. The adoption of non-conventional notation for flow diagrams has the inherent disadvantage of not being readily understood by most people, coupled with the fact that there is a steep learning curve for users. The program is constructed by creating diagrams with special symbols, followed by the addition of instructions into them. Fig. 2 shows a diagram for creating assembly language codes in an existing software tool 300. The diagram is primarily made up of nodes 301, control flow wires 302, and data flow wires 303. Each node 301 represents a set of instructions which are embedded in a text file. Alternatively, nodes 301 can be further expanded using the node expansion operator 304 to form expansion nodes 305 which can contain another diagram. Symbol 306 is used to denote the common entry and exit point of the expansion node 305. Control flow wires 302 determine the flow of control or program execution. Data flow wires 303 determine the assignment and manipulation of values. The data types of these values are indicated by identifiers 307. Node 308 represents a parameter passing mechanism. It is used to input parameter values into node 301. Node 309 is an iterative node that can have multiple inner rings. The inner ring 310 becomes active as the program flow departs from it to execute code in node 301. When program flow re-enters node 309, execution begins from the same inner ring 310. The fully constructed diagram is then converted into assembly language codes. When program errors in the codes are discovered by the programmer or highlighted by the assembler, changes have to be made to the diagram, rather than the code text. This is tedious and cumbersome. In addition, the tool 300 is incapable of converting the assembly codes back into diagrams. Therefore, it is unable to process any existing assembly codes.

The above mentioned CASE tools are limited in their scope and extent of use. No CASE tool is currently available that integrates the generation of editable, hierarchical flow charts from existing assembly codes and the conversion of the flow charts back into assembly codes.

This Sheet Has Been Replaced



Summary of the Invention

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

an assembly code editing window sub-system;

an assembly code to flow chart conversion sub-system;

a flow chart editing window sub-system;

a flow chart navigator window sub-system;

a flow chart to assembly code conversion sub-system;

an external assembler interface sub-system;

a status window sub-system;

an assembly language library sub-system;

an assembly language translator engine sub-system; and

a system management sub-system.

The assembly code editing window sub-system consists of a means to allow assembly codes to be displayed as text, a means to perform conventional text editing operations, a means to differentiate between distinct segments of code and the various mnemonics with the use of colour syntax highlighting for improved readability, a means to dynamically alter the colour of text, according to its meaning, as it is being typed into the editor, a means to display text in user selectable fonts, a means to search and replace specific text.

The assembly code to flow chart conversion sub-system consists of a means to convert assembly codes, which can contain constructs of a commonly used language, such as the C language, into hierarchical flow charts, a means to parse and examine the codes to convert them into flow chart shapes, a means to link successive flow chart shapes by inserting connecting arrows, a means to connect branch shapes with their target labels by inserting branch arrows, a means to allow logical blocks of code to be grouped into modules and sub-modules by using pre-defined heuristics to give a

This Sheet Has Been Replaced



hierarchical representation of the entire program, a means to ensure that the converted flow chart will be logical and understandable by using assembly code integrity checking mechanisms.

The flow chart editing window sub-system consists of a means to display a hierarchical flow chart, a means to add new shapes and arrows to the flow chart, a means to edit or delete existing shapes and arrows, a means to insert assembly codes in the form of shapes and arrows into the flow chart, a means to insert assembly codes in the form of text into the flow chart by automatically converting the text into shapes and arrows, a means to automatically insert branch arrows which begin from a branch shape and end at the target label, a means to differentiate the various flow chart shapes by their border and background colour, a means to facilitate the alignment of shapes and arrows by using the snap function, a means to efficiently locate specific flow chart shapes, a means to assign shape line numbers to every flow chart shape, a means to automatically reassign shape line numbers when the flow chart is edited, a means to edit text within the flow chart shapes and their associated comments by using the in-place-editing function, a means to automatically append the target label's line number to the branch shape's text; a means to automatically refresh the target label line number in the branch shape's text when the label is changed.

The flow chart navigator window sub-system consists of a means of presenting the hierarchical flow chart in the form of an interactive tree structure that allows easy access to all the shapes, a means of navigating among the tree nodes and branches using the mouse or keyboard, a means to expose and hide the flow chart layers by expanding and collapsing operations such that only the appropriate nodes in the branches are shown and the unnecessary ones hidden, a means to synchronise the navigator window's contents with those in the flow chart editing window sub-system, a means to highlight the corresponding shape in the flow chart editing window when a node in the tree is selected, a means to efficiently locate flow chart nodes by specifying their text contents, a means to copy, cut or move nodes from one location or layer to another by performing drag and drop operations with the mouse, a means to differentiate between the various categories of nodes by placing different image icons beside them, a means to edit node text by using the in-place-editing function, a

This Sheet Has Been Replaced



This Sheet Has Been Replaced



means to notify the flow chart editing window about modifications that are made to the tree structure or node contents in the navigator window, so that both the windows' data are synchronised and up-to-date.

The flow chart to assembly code conversion sub-system consists of a means to convert the edited flow chart back to assembly codes, which can contain constructs of a commonly used language, such as the C language, a means to intelligently track the course of the flow chart and examine its hierarchical structure to generate the assembly codes, a means to identify and convert certain flow chart structures into C language constructs, a means to combine the instruction, label and comments into a line of text in the assembly code file, a means to channel the text output to a new file or to overwrite the text in the original assembly code file, a means to perform a check on the correctness of the flow chart structure and shape contents before conversion.

The external assembler interface sub-system consists of a means to couple assemblers to the system so that compilation can be done from within the system, a means to select a particular assembler for use, a means to specify command line options for text-based assemblers, a means to allow messages generated by the assembler to be channelled to the status window sub-system or saved as a text file.

The status window sub-system consists of a means to display information about the current state of the system, a means for the system to report errors and warnings encountered during the course of code development, a means to enable the external assembler interface sub-system to report output messages from the assembler, a means to allow both the assembly code to flow chart conversion and the flow chart to assembly code conversion sub-systems to report progress, status and error messages.

The assembly language library sub-system consists of a means to store the assembly mnemonics and syntax of several types of assembly languages, a means to store C language constructs and syntax definitions if they are provided by the language, a means to categorise the mnemonics according to their function, a means to map the mnemonic categories to specific flow chart shapes, a means to allow the creation, editing or deletion of assembly mnemonics, a means to specify the assembly language syntax structure, a means to edit the C language constructs and syntax

definitions, a means to select an assembly language to be used in the development process.

The assembly language translator engine sub-system consists of a means to convert codes from one assembly language to another, with provisions for C language constructs, a means to convert codes in one assembly language into a flow chart in another assembly language, a means to convert a flow chart in one assembly language to codes in another assembly language, a means to detect codes which cannot be converted and to highlight them to the programmer for manual conversion, a means for the programmer to specify and load both the source and destination assembly languages.

The system management sub-system consists of a means to ensure that the distributed information throughout all the windows are synchronised and up-to-date, a means to determine the presence and validity of the relevant windows before any conversion process begins, a means to automatically detect changes made to assembly code files, a means to store the list of assembly code file modifications in a register, a means to print the assembly codes and flow chart, a means to allow programmers to customise the window environment and system settings, a means to assist programmers in using and understanding the system by providing a help facility.

Brief Description of Drawings

Fig. 1 is a screen layout of an existing non-editable, single layer flow chart generator system.

Fig. 2 is a diagram containing proprietary symbols used by an existing software tool to create assembly language codes.

Fig. 3 is a schematic block diagram of the system showing the interaction between the various sub-systems, including the path of assembly code and flow chart data, according to an embodiment of the invention;

Fig. 4 is a screen layout of the system displaying an assembly code file and its corresponding flow chart and navigator windows.

This Sheet Has Been Replaced

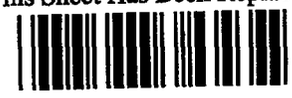


Fig. 5 is a schematic block diagram of the work flow model illustrating the path data takes during a typical instance of use.

Detailed Description of the Best Mode of Performing the Invention

Fig. 3 shows the internal architecture of a graphical microcontroller software development system 10. The system is made up of the following :

an assembly code editing window sub-system 20;

an assembly code to flow chart conversion sub-system 25;

a flow chart editing window sub-system 30;

a flow chart navigator window sub-system 35;

a flow chart to assembly code conversion sub-system 40;

an external assembler interface sub-system 45;

a status window sub-system 50;

an assembly language library sub-system 55;

an assembly language translator engine sub-system 60; and

a system management sub-system 15.

This Sheet Has Been Replaced



The assembly code editing window sub-system 20 comprises a text editor sub-system 21 and a colour syntax highlighting engine sub-system 22.

The text editor sub-system 21 supports conventional text editing operations such as text entry, deletion, copying, cutting and pasting. It is capable of handling very large text files. Each window 20a displays the contents of one assembly code file. An unlimited number of text editor windows 20a can be opened simultaneously, solely determined by the available system resources. The text font and window background colour can be changed and tailored to suit individual preferences. These settings can be stored and retrieved from the system manager sub-system 15. Useful features such as the search and replacement of specific text are included and are also handled by the text editor sub-system 21.

The colour syntax highlighting engine sub-system 22 enhances the readability and the presentation of assembly codes in the text editor window 20. It detects the different classes of assembly mnemonics and presents each class in user defined colours. It can also identify program segments which are delineated by blocks of comments and similarly display them in different colours. As the text is edited, the colour of the word being typed changes dynamically according to its function. This feature helps the programmer to detect syntax errors while typing.

To differentiate between the many text editor windows 20a, various unique colours are assigned to the caption bar of the windows 20a in addition to displaying the assembly code filenames. When flow charts are generated, the flow chart editing window 30a caption bars will likewise adopt the same colours as their corresponding text editor windows 20a. In this way, it is easy to associate a text editor window 20a with its associated flow chart editing window 30a. The text editor windows 20a conform to the established Windows user interface guidelines, allowing it to be cascaded, tiled and arranged within the system 10.

The assembly code to flow chart conversion sub-system 25, also known as the front end 101, converts assembly codes, which can contain constructs of commonly used languages, such as the C language, into a hierarchical flow chart. The user initially selects a text file containing the assembly code for input into the system 10. Internally, the codes are parsed and examined for label, instruction and relevant data content. This information is then used to construct the flow chart. Each line of code is assigned a flow chart shape according to its function. The mapping of the mnemonic to its corresponding shape is determined by the assembly language library sub-system 55. 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 arrows 86b are inserted between the shapes as they are created. Branch lines 87b are added to join branch instruction shapes 82b with their target labels 80b. Line numbers 88 and comments 89 that accompany each instruction are conditionally drawn depending upon the user preference settings. The interpretation of the assembly mnemonics and syntax is determined by the assembly language chosen.

This Sheet Has Been Replaced



Using certain pre-defined heuristics, logical blocks of code are grouped into functions, which may in turn be further grouped into sub-modules and modules. This layered approach gives the user a hierarchical overview of the entire program. Therefore, at the upper levels, a flow chart shape may represent a collection of codes beneath it. The different levels in the flow chart may be accessed by using the flow chart navigator window 35a.

Assembly code integrity checking mechanisms are installed into the conversion sub-system 25. These are used to ensure that the flow chart created will be logical and understandable. Instances of missing or duplicate labels, convoluted code and calls without returns, amongst others, are highlighted to the programmer via the status window sub-system 50. The sub-system 25 is invoked by pressing the assembly code to flow chart conversion icon 25a.

Once the flow chart has been created, the flow chart navigator window sub-system 35 receives the entire flow chart hierarchy structure and shape details from the flow chart editing window 30a. The flow chart navigator window 35a uses this information to derive and draw the tree structure and nodes.

The flow chart editing window sub-system 30 is responsible for displaying the flow chart. It comprises a flow chart drawing engine sub-system 31, a flow chart object management sub-system 32 and a flow chart line number generator sub-system 33.

Flow chart shapes generated by the assembly code to flow chart conversion sub-system 25 are displayed in the flow chart editing window 30a. The flow chart drawing engine sub-system 31 handles the algorithms for drawing and adding new flow chart shapes. There are six types of flow chart shapes namely, label 80b, non-decision 81b, decision 82b, call 83b, return 84b and comment 85b shapes. These flow chart shapes can be easily distinguished from each other by their unique contour, border and background colour. The drawing engine sub-system 31 also controls the drawing of shape connecting arrows 86b, branch arrows 87b, line numbers 88, comments 89 and grid lines. Branch arrows 87b are constructed such that they do not overlap each other unnecessarily. They change their line colour whenever they are selected, to make the identification of a particular branch line easier. Shape and arrow

This Sheet Has Been Replaced



editing operations include dragging, cutting, pasting, copying, repositioning and resizing. A snap facility is provided, which forces the vertices of the shapes to conform to the boundaries of the grid lines. This feature facilitates the alignment of shapes.

Different levels in the flow chart can be accessed from within the flow chart editor window 30a by right clicking the mouse on the shape concerned. This produces a floating context menu with options available for the programmer to go up or down a level. After which, the window 30a refreshes itself to display the new flow chart at the chosen level.

The mnemonic and associated parameters that each flow chart shape represents are drawn as text within the shape's boundaries. Comments 89, if any, are drawn to the right of the shape. Text can be edited from within the shape itself. The editing mode can be invoked by double clicking on the flow chart shape concerned with the left mouse button. This produces an in-place-edit box, where changes can subsequently be made to the code. Comments 89 are edited in the same manner.

The flow chart object management sub-system 32 is responsible for the creation and deletion of flow chart objects. There are several ways of adding additional flow chart objects at any point in the existing flow chart. They include the normal drawing method and the automated insertion method.

In the normal drawing method, new flow chart objects and arrows can be drawn by first selecting the appropriate flow chart shape and arrow icons from the system 10. There are six types of flow chart shape icons namely, label 80a, non-decision 81a, decision 82a, call 83a, return 84a and comment 85a icons. There are two types of arrow icons namely connecting arrow 86a and branch arrow 87a icons. The mouse is then used to specify its position and size. Thereafter, text is entered into the shape by using the in-place-editing function. Connecting 86b and branch 87b arrows have to be drawn manually.

The automated insertion of shapes is accomplished by either typing in text or by copying and pasting groups of connected flow chart shapes or modules. The text entered will be converted into the appropriate flow chart shapes with connecting

This Sheet Has Been Replaced



arrows 86b in between before it is inserted into the flow chart. The programmer has to specify the point of insertion.

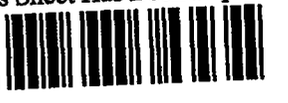
All flow chart shapes have line numbers 88 drawn on their left. Line numbers 88 are arranged in ascending order, starting from the first flow chart shape at the top. The flow chart line number generator sub-system 33 utilises an intelligent method to dynamically assign line numbers 88 to flow chart shapes across the different layers. The line numbers 88 generated have a direct correlation with line numbers in the assembly code generated from the flow chart. The addition and removal of flow chart shapes or modules will invoke the line number generator 33, to reassign line numbers. For shapes that are present in the lower layers, line numbers 88 are assigned to them according to the order in which they will appear in the text when the flow chart is converted. The sub-system 33 is invoked by pressing the flow chart line number generator icon 33a.

Branch instruction shapes 82b have their target label's line numbers 90 appended to their text to facilitate the tracing of program flow. To accomplish this, the flow chart object management sub-system 32 locates the target label from its internal database to retrieve the line number 88. This number is then add to the branch shape's 82b text. If the target label 80b of a branch shape 82b is changed, the branch arrow 87b that emerges from the shape 82b will then automatically point towards the new target label 80b. Similarly, if the branch arrow's 87b destination is changed from one label 80b to another, the appended line number will change automatically.

The flow chart navigator window sub-system 35 provides quick access and search of particular flow chart shapes in the flow chart editing window 30a. It comprises a tree structure drawing engine sub-system 36, a tree object management sub-system 37 and a flow chart object synchronisation sub-system 38.

The tree structure drawing engine sub-system 36 handles the algorithms for drawing the tree, whose nodes are composed of both text 97 and images 96 associated with the flow chart shapes. The image 96 drawn depends upon the shape type. The image is a smaller version of the actual shape's outlook in the flow chart. The sub-system 36 also handles changes in the tree structure appearance when its branches are expanded or contracted in order to display the desired flow chart shapes or to hide the

This Sheet Has Been Replaced



unnecessary ones. A node that can be further expanded to produce a tree branch has a 'plus' sign image added to it. Conversely, a 'minus' sign image 95 is used to denote that a current node's branch has been expanded. The text 97 associated with a node can be edited using the in-place-editing function. If the text 97 is revised to change the nature of its function, the image 96 will change automatically. The mapping of the mnemonic to the correct image 96 is derived from the assembly language library sub-system 55.

The tree object management sub-system 37 maintains a list of currently existing flow chart shapes and arrows. It is responsible for the creation and deletion of tree nodes, including algorithms for handling the copying, cutting and pasting of tree nodes from one location or layer to another and for keeping track of the hierarchical relationships among the flow chart shapes. Information pertaining to the nodes are stored in a database within the sub-system 37. This information includes the node's text, image type, hierarchical level and its state and position in the tree.

The flow chart object synchronisation sub-system 38 keeps data in the various windows up-to-date. When a node is selected from the tree in the navigator window 35a, the flow chart editing window 30a is notified. It reacts by searching for the node's corresponding flow chart shape and automatically highlights it. Likewise, when a selection is made on a shape in the flow chart editing window 30a, the flow chart navigator window 35a is informed, and will search through its tree node list and highlight the appropriate node. If the node resides on an unexpanded branch, the branch will be expanded to reveal the node concerned. In this way, the flow chart drawing will always reflect the current state of the tree.

Drag and drop operations in the flow chart navigator window 35a also affect the contents and arrangement of shapes in the flow chart editing window 30a. The navigator window 35a communicates any flow chart structure and node changes to the flow chart editing window 30a, and vice versa. Both windows refresh themselves after new information is received. The flow chart object management sub-system 32 keeps track of the multiple layers of the flow chart. While traversing between layers, the flow chart editing window 30a refreshes itself to display a new flow chart at a different level.

This Sheet Has Been Replaced



The flow chart to assembly code conversion sub-system 40, also known as the back end 105, is responsible for converting the edited flow chart into assembly codes, which can contain constructs of commonly used languages, such as the C language. It achieves this by intelligently tracking the course of the vertical 86b and branch 87b line arrows through the flow chart, and by examining its hierarchical structure. To determine the connectivity between an arrow and a shape, the sub-system 40 expands the area covered by both objects by a pre-determined amount. If the areas overlap each other, the arrow and the shape are said to be connected. The location of the arrow head is then sought to ascertain whether program flow is directed into or out of the shape concerned. This operation is conducted for all the shapes and arrows in every layer of the flow chart.

– The sub-system 40 is also capable of identifying and converting certain flow chart structures into their equivalent C language constructs. The relevant instructions and labels are combined together with their comments to form a line of text in the assembly code file. The programmer may choose to overwrite the original text in the assembly code editing window, labelled as 'Input Code' 20a or create a new assembly code editing window, labelled as 'Output Code' 20b. Before any text is generated, the conversion engine ascertains whether the flow chart's structure and shapes are sound and correct. Errors may arise from unconnected shapes and arrows or wrongly constructed modules. The programmer is notified of any anomalies detected by the sub-system 40 via the status window 50a. The sub-system 40 is invoked by pressing the flow chart to assembly code conversion icon 40a.

The integrated development environment concept adopted by the system 10 not only enables full text and flow chart editing facilities, it also couples itself to assemblers through the external assembler interface sub-system 45. The system 10 maintains flexibility by allowing the user to choose the appropriate assembler type to compile the codes. The assembler selection dialog box presents the list of available assemblers for a particular assembly language. This list can be edited. The sub-system stores the path and executable file names of the assemblers, including the command line options available for each assembler type. If the assembler is a text-based program, the command line options that are required can be specified. The assembler

This Sheet Has Been Replaced



can be invoked from within the system 10 and the relevant codes channelled to it for compilation. The status window 50a displays messages that are produced by the assembler during the course of compilation as well as the success, error or warning notes at the end of the compilation. Therefore, the programmer does not need to leave the system 10 for code compilation. For review purposes, the series of messages generated can be saved to a file or printed.

The status window sub-system 50 provides information about the state of the system 10. It reports status, error and warning messages encountered during the conversion process, the assembly code and flow chart editing operations as well as the assembler output messages. The status window 50a contains a set of pre-defined functions that other sub-systems can use to output messages.

The assembly language library sub-system 55 is divided into the library management 56 and language selection sub-systems 57.

The library management sub-system 56 contains a collection of assembly mnemonics and syntax structure definitions from various commonly used assembly language families. It also contains C language constructs and syntax definitions, if they are provided by the language. Each language family's definitions are stored as a file, which is loaded into the system 10 when required. The mnemonics are grouped into seven categories namely, decision, non-decision, comments, labels, assembler directives, calls and returns. Each category is associated with a particular flow chart shape. The shapes adopted by the system 10 conform to internationally recognised symbols for flow charts. Every mnemonic has its own syntax definition, to enable the code parser to extract the necessary mnemonic parameter information correctly. A built-in editor allows both the mnemonic name, syntax definition and the category it belongs to be edited, including the C language constructs and syntax definition. The library is expandable to accommodate new languages, which can be added by using the editor.

The assembly language selection sub-system 57 provides a means for the programmer to choose the assembly language to be used. This loads the language's mnemonics and syntax definitions into the system 10. This selection is done prior to converting the assembly codes into the flow chart format. During the code parsing

This Sheet Has Been Replaced

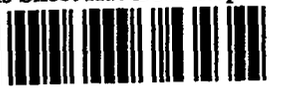


stage of the assembly code to flow chart conversion process 25, every instruction is matched against the previously loaded set of mnemonics to determine the category with which it belongs to. The appropriate shape is then assigned to it. In the assembler language translator 60 mode, both the source and target assembly languages are loaded into the system 10.

The conversion of code from one assembly language to another is accomplished by using the assembly language translator sub-system 60. Assembly mnemonics from the source language are mapped to their corresponding mnemonics from the destination language. One mnemonic from the source language may be represented by either one or several mnemonics from the destination language. The mnemonic mapping relationships between any two languages are stored in a dictionary. The differences in syntax between the two languages has to be taken into account, such that both the original and the translated code have similar meanings. The segments of code which have no equivalent in the destination language are highlighted and its translation is left to the programmer. This sub-system 60 ignores assembler directives and other language specific commands. In addition, codes in one assembly language can be converted into a flow chart in another assembly language, and a flow chart in one assembly language can be converted to codes in another language.

The system manager sub-system 15 forms the back bone of the entire system 10. Fig. 4 shows the screen layout of the system 10. The system manager 15 appears as the shell 15a wherein all the various windows are contained. It performs several essential functions. It ensures that the flow chart structure and shape data are correctly synchronised throughout all the windows concerned. When a new shape is added to the flow chart editing window 30a, its corresponding node must be added to the tree in the flow chart navigator window 35a. Likewise, drag and drop operations in the navigator window 35a that result in structural changes must be reflected in the flow chart editing window 30a. These updating operations are especially critical when multiple code files are simultaneously present within in the system 10. The system 10 also supports the copying, cutting and pasting of shapes across the various flow chart windows 30a by using the drag and drop function. The system manager 15 also

This Sheet Has Been Replaced



includes the facility to automatically detect changes made to assembly code files. The series of code file modifications together with the author's name and time of edit, are stored in a register for reference.

The system manager sub-system 15 also ascertains the presence and validity of the relevant windows before any conversion process begins. An assembly code to flow chart conversion 25 would not be possible if the assembly code files were not previously chosen and loaded into the system 10. The printing of the flow chart and text is handled by the system manager 15. A screen preview of the flow chart or text to be printed is included to allow the programmer to assess its appearance on paper before printing.

The parameters controlling the appearance of both the assembly code 20a and flow chart editor windows 30a can be modified and stored by the user. These settings include the colours, sizes, text font and inter-shape spacing of the flow chart shapes, the font and background colour of the text editor 20a, the assembly language and assembler selection settings. Default settings are pre-loaded into the system 10. The user can modify these settings based on his personal preference. More than one setting can be stored. When the system is reused at a later date, the user's previous settings are automatically loaded.

The work flow model 110 of the system 10 describes the path data takes during a typical instance of use. It is divided into the front end 101 and back end 105, with the flow chart and code editing phase 103 in the middle, as shown in Fig. 5.

The work flow 110 starts with the loading of the selected assembly code file 100 into the code editing window 20a. The front end 101 follows with the conversion of assembly codes into a hierarchical flow chart 102, which is displayed in the flow chart editing window 30a. Thereafter, the flow chart is edited. Various shapes can be added, modified or deleted, including functions and modules. This forms the flow chart and code editing phase 103. When editing is completed, the back end 105 converts the edited flow chart 104 into assembly codes 106. These assembly codes 106 may either replace the text contents of the original assembly code file 100, or be written to a newly created file. The assembler is then invoked to compile the codes.

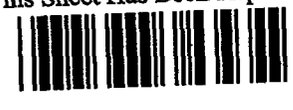
This Sheet Has Been Replaced



In addition, a new set of assembly codes can be created by constructing a new flow chart. This encourages the user to adopt modular programming techniques. The codes are produced by invoking the back end 105.

The system work flow model 110 described above depicts the situation whereby only one code file has to be edited. The system 10 is also capable of handling multiple assembly code files 100 as input. In this case, every code editing window 20a will have its associated flow chart editing 30a and navigating 35a windows. A code file's group of windows will have a unique caption identifier and colour to distinguish itself from other groups of windows. The system manager 15 co-ordinates the entire system 10, ensuring that the data and window states are up-to-date and synchronised.

This Sheet Has Been Replaced



Claims

1. An integrated system for the development of assembly language codes for microcontrollers, comprising :

an assembly code editing window sub-system;

an assembly code to flow chart conversion sub-system;

a flow chart editing window sub-system;

a flow chart navigator window sub-system;

a flow chart to assembly code conversion sub-system;

an external assembler interface sub-system;

a status window sub-system;

an assembly language library sub-system;

an assembly language translator engine sub-system; and

a system management sub-system.

2. A sub-system according to claim 1, wherein the said assembly code editing window sub-system comprises :

* C L M S *

This Sheet Has Been Replaced

a text editor sub-system; and

a colour syntax highlighting engine sub-system.

3. A sub-system according to claim 2, wherein the said text editor sub-system supports conventional text editing operations on very large text files, user-definable fonts and window background colour, including text search and replacement capabilities.

4. A sub-system according to claim 2, wherein the said colour syntax highlighting engine sub-system detects the different mnemonics and program segments and presents them in various colours for differentiation and enhanced readability.

5. A sub-system according to claim 1, wherein the said assembly code to flow chart conversion sub-system comprises of intelligent methods for parsing the assembly codes which can contain C language constructs, to extract assembly mnemonics and to detect program structures and flow in order to generate the editable hierarchical flow chart.

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

a drawing engine sub-system;

a flow chart object management sub-system; and

a flow chart line number generator sub-system.



7. A sub-system according to claim 6, wherein the said drawing engine sub-system handles the algorithms for the drawing, re-positioning and re-sizing of the various flow chart shapes, associated comments, line numbers, connecting and branch arrows, including the editing of text within them.

This Sheet Has Been Replaced



This Sheet Has Been Replaced

8. A sub-system according to claim 6, wherein the said flow chart object management sub-system is responsible for the creation and deletion of flow chart shapes, including algorithms for handling the insertion of new shapes or assembly codes into an existing flow chart, and for keeping track of the shapes in the flow chart layers.

9. A sub-system according to claim 6, wherein the said flow chart line number generator sub-system utilises an intelligent method to dynamically assign line numbers to flow chart shapes across the layers even while editing is in progress, and these numbers have a direct correlation to the line numbers in the assembly code that is generated from the flow chart.

10. A sub-system according to claim 1, wherein the said flow chart navigator window sub-system comprises :

a tree structure drawing engine sub-system;

a tree object management sub-system; and

a flow chart object synchronisation sub-system.

11. A sub-system according to claim 10, wherein the said tree structure drawing engine sub-system handles the algorithms for drawing the tree containing nodes composed of text and images which are associated with the flow chart objects, the expanding and collapsing of the tree branches to display and hide nodes held within them, and the drag and drop function to copy, cut or paste nodes, and the editing of node text.

12. A sub-system according to claim 10, wherein the said tree object management sub-system is responsible for the creation and deletion of tree nodes, including algorithms for handling the copying, cutting and pasting of nodes and for keeping track of the hierarchical relationships among them.

* C L M S *

This Sheet Has Been Replaced

13. A sub-system according to claim 10, wherein the said flow chart object synchronisation sub-system provides quick navigation and search for flow chart shapes by highlighting the desired shape in the corresponding flow chart editing window when a node in the tree is selected.

14. A sub-system according to claim 1, wherein the said flow chart to assembly code conversion sub-system comprises of intelligent methods to detect flow chart shape connectivity and hierarchical structure to generate assembly codes, which can contain C language constructs.

15. A sub-system according to claim 1, wherein the said external assembler interface sub-system allows the selection of different assembler types, the specification of command line options for text-based assemblers, the ability to invoke the desired assembler from within the system and the channelling of assembler output messages to the status window sub-system.

16. A sub-system according to claim 1, wherein the said status window sub-system supplies information regarding the state of the conversion processes, the state of the assembly code and flow chart editing operations and the assembler output messages.

17. A sub-system according to claim 1, wherein the said assembly language library sub-system comprises :

a library management sub-system; and

an assembly language selection sub-system.

* C L M S *

18. A sub-system according to claim 17, wherein the said library management sub-system contains collections of assembly mnemonics, syntax definitions and C language constructs from various commonly used assembly

* C L M S *

languages which can be edited and it also has facilities for the addition of new languages into the system.

19. A sub-system according to claim 17, wherein the said assembly language selection sub-system comprises a selection interface that allows the programmer to choose the desired assembly language to be used, and the corresponding collection of assembly mnemonics, syntax definitions and C language constructs are then loaded into the two conversion sub-systems so that the assembly codes can be correctly parsed and converted into flow chart shapes, and vice versa, including the parsing of codes to be inserted into the existing flow chart.

20. A sub-system according to claim 1, wherein the said assembly language translator sub-system contains algorithms for converting code from one assembly language to another, or to convert code from one assembly language to a flow chart in another language, or to convert a flow chart in one assembly language to code in another language, using a dictionary which stores the mapping relationships of mnemonics between two languages.

21. A sub-system according to claim 1, wherein the said system management sub-system ensures that the various windows and data are correctly synchronised together during the conversion and editing stages, so that when multiple code editing windows reside in the system, they correlate to the correct flow chart editing window, to handle the printing of flow charts and codes, to store user preference settings, to detect changes made to code files and store these modifications into a register.

22. An integrated system for the editing and compilation of assembly codes which comprises an assembly code editing window sub-system similar to the sub-system in claim 2, an external assembler interface sub-system similar to the sub-system in claim 15, an assembly language library sub-system similar to the sub-

This Sheet Has Been Replaced

system in claim 17 and a status window sub-system similar to the sub-system in claim 16.

23. An integrated system for the generation of editable and hierarchical flow charts from assembly codes which comprises an assembly code editing window sub-system similar to the sub-system in claim 2, an assembly code to flow chart conversion sub-system similar to the sub-system in claim 5, a flow chart editing window sub-system similar to the sub-system in claim 6, an assembly language library sub-system similar to the sub-system in claim 17 and a status window sub-system similar to the sub-system in claim 16.

24. An integrated system for the generation of assembly codes from hierarchical flow charts which comprises a flow chart editing window sub-system similar to the sub-system in claim 6, a flow chart to assembly code conversion sub-system similar to the sub-system in claim 14, an assembly code editing window sub-system similar to the sub-system in claim 2, an assembly language library sub-system similar to the sub-system in claim 17 and a status window sub-system similar to the sub-system in claim 16.

25. An integrated system for the translation of codes from one assembly language to another which comprises an assembly code editing sub-system similar to the sub-system in claim 2, an assembly language translator sub-system similar to the sub-system in claim 20, an assembly language library sub-system similar to the sub-system in claim 17 and a status window sub-system similar to the sub-system in claim 16.

* C L M S *

This Sheet Has Been Replaced