METHOD FOR LOADING AN APPLICATION PROGRAM IN A CHIP CARD

The purpose of this invention is to load an application program in a portable chip support, particularly chip cards. It is used in situations where we want to have several applications of different types on the same chip simultaneously. "Application" refers to the operations which occur within a chip card and within a card reader in which the card holder inserts the card to meet a certain need. These operations include electronic processing within the card and card reader circuits and the actions – some of which may be mechanical – on the peripherals to which the card reader is linked. An application thus includes a program for card operation stored in the card memory and the program contains a set of instructions.

The general structure of a chip card includes a microprocessor in contact with a program memory, one or several data memory units and an interface for communication with the outside world. It was thought that putting several application programs on the same card could make the use of chip cards more practical. For example, a bank program and a car reservation program could be put together. The two service providers responsible for these two application programs, a bank and a banking organisation would generally have nothing in common and would be completely independent of each other. They may not even know that the chip cards which their client have also include application programs for other service providers.

The main problem with this type of sharing of chip card resources among several applications is the mutual respecting of the integrity of the data and instructions of the various application programs.

To solve the problems due to this co-existence, several approaches were developed based on the principle of intervention by a higher authority, such as the company which manufactures the cards. Several approaches would be possible.

For example, manufacturers of the integrated circuits which are built into the cards could have this authority. Otherwise, the integrated circuit manufactured can be totally blank or may include a set

of basic instructions to operate the card. This set of basic instructions is the operating system of the integrated circuit. This system can be programmed within the integrated circuits, either directly at the time of manufacturing (in this case it is usually programmed with a template), or during a test operation.

Another possibility is that the operating system could be installed in the integrated circuit by a company which gives the service provider a batch of cards with an operating system and application programs adapted to his needs. In this latter case, this co-existence of various applications on the same integrated circuit in the same chip card would be achieved by designing, in each case, a special operating system so that several different (but pre-identified) applications could be stored on the card without one application affecting the others unless the various service providers involved agree on this.

The drawback of this approach is that the company which uses it must know all of the intended uses for the cards for the service providers from the outset. This would make revisions and upgrades difficult. Another drawback is that this approach depends on the degree of confidence which the various service providers have in the company which installs the operating system and application programs in the integrated circuits of the cards. As these applications

include actions which are sensitive for the service providers, this confidence may be hard to establish and to maintain.

In addition to the problem of upgrading the applications, there is also the difficulty of storing the application programs in the memory of the integrated circuit of the chip card because this must be done by designating absolute addresses for the memory areas available on the card, i.e. it would be necessary to divulge the entire structure of the integrated circuit of the card, particularly divulging the addresses where the card secrets are stored. These private keys for coding must be known in order to store application programs on the chip card.

The invention solves this problem by including a loading macro instruction in the operating system of the integrated circuits which calculates the absolute addresses where the application program instructions must be stored based on a known address, and which modifies the application program as a result and records the instructions for this program at the calculated addresses.

In the invention, the integrated circuit's application program also includes a definition of a data memory space which will be necessary for this use. The invention's loading macro instruction reads this definition which is present in the application program (generally at the beginning of the program), and

reserves in the data memory the space needed to store the corresponding data. During loading, the invention's loading macro instruction also updates, ideally, a memory area which represents the storage in memory of the application program for this application. This memory area may include a file descriptor, for example.

The invention thus relates to method for loading an electronic chip on a portable chip support which includes a microprocessor and memory unit with an program for operation of this chip card. The program includes instructions in which:

- the chip is provided with an operating system which includes a program for loading the memory. This operating system controls the operation of the microprocessor and the memory,

- using the memory loading program, the application program is recorded in the memory,

characterised in that:

- in the application program we define a quantity of space needed in the chip memory to contain the data to be used by this application program is defined,

- during loading, this memory loading program calculates the absolute addresses to record the data used in the application program and the instructions for the application program in the memory,

- the application program is modified accordingly

- this loading program records the data and instructions present in the application program at the absolute addresses calculated.

Another problem inherent to chip cards is giving service providers the possibility of programming new applications or modifying their applications without having to know the functioning of the invention's operating system. In the invention therefore, the integrated circuit of the card will include a command interpreting program. Such a command interpreting program can have the microprocessor of the integrated circuit execute instructions in a high-level language (symbolic) which are read in a program memory and directly interpreted on the fly, i.e. transformed into instructions which can be executed by the microprocessor. These executable instructions can then be loaded as is into the microprocessor's instruction register.

In this case, the only constraints affecting the service providers are that they must provide their application programs with a definition of the space needed for the data and that they must use a program in a high-level language which is commonly used in computing. This high-level language could be C, FORTH, FORTRAN or COBOL or yet another. If necessary, the loading macro instruction could first recognise the language being used in order to properly interpret the

commands. There would thus be different interpreters for different languages.

The object of this, invention therefore is a method like the previous one characterised by the following:

- the chip is provided with an operating system including a command interpreter program 33,

- using the memory loading program the memory of the chip records an application program 15 for using of the chip card including instructions, written in a high-level language, which cannot be directly executed by the microprocessor, for the purpose of having the microprocessor execute the interpreted instructions which result from these instructions (which cannot be directly executed) once they have undergone interpretation by the command interpreter program,

- at the time of this recording, the memory loading program calculates the addresses to record the program instructions in a high-level language into the chip memory,

- the high-level language instructions based on this calculation, are modified

- these high-level language instructions are recorded in the memory at absolute addresses calculated by the loading program.

The invention will be better understood with the following description and the accompanying figures. These are only provided as an indication and are in no

way limiting with regard to the invention. These figures show:

- figure 1: structure of an integrated circuit for an electronic support suitable for using the invention method;

- figure 2: a flow chart showing the sequence of operations for loading according to the invention method;

- figures 3a to 3c: a schematic example of a program written in a high-level language with relative addressing, and the various transformations which it undergoes before it is recorded in the memory of the integrated circuit;

- figure 4: a description of the syntax of certain instructions which can be used in such a program.

Figure 1 shows the structure of an integrated circuit for a portable support which could use the invention method. An electronic chip 1 is held by a portable support which is not shown. In an example, the portable support is a chip card. The chip 1 includes a microprocessor (2 and a memory. In the example, the memory is a double memory. It includes a first memory 3 which is for storing an operating system for the integrated circuit. The memory 3 is preferably a non-programmable memory unit, programmed by template for example or possibly programmable but then locked after its programming so that it can no longer be programmed. It is a non-volatile memory, the memory cells could be

either EPROM or EEPROM type. Another part 4 of the memory can be programmed and erased at will and is preferably of the EPROM type. It is also non-volatile. The service provider can include here all of the application programs which he wants to put in so that the integrated circuit 1 can handle all of the desired applications.

The operating system recorded in the memory 3 includes at least a first set of instructions 5 of a known type. For example, these instructions could be MOVE, MULT, PLUS or GOTO. Once they are loaded into the instruction register 6 of the processor 2, their purpose will be to trigger the transfer of data (or instructions) into a data register 7 of the microprocessor 2, so that they can be processed by the microprocessor. The functioning of this system (known type) is as follows. When the application program (contained in part 6 of the memory 4 is started, an instruction read in this part triggers the loading of an instruction from the operating system 5 into the register 6 (in some cases it triggers the successive loading of several instructions from the operating system). At the same time, the useful data are loaded into the data register 7 of the microprocessor. The data may for example be taken from one part 9 of the memory 4. Then these data are processed by the microprocessor 2, with the result then going to the register 7 (for example). The parts 8 and 9 together

make up an area 10 of the memory 4 reserved for the application in question. In the example shown, the application is schematically a banking-type application. It could for example correspond to the method of viewing a bank account balance on a card reader screen.

In the state of the art, the loading of the instructions for the banking application and the data concerning this application in areas 8 and 9 respectively in the memory 4 is triggered by an external system 11 which supersedes the integrated circuit 1 through an interface 12 for communication of the integrated circuit 1 with the outside world. The interface 12 and the chip circuits are linked to each other by a bus 13 which, to simplify things, will be used to transmit addresses, data and commands.

The driving of a write circuit 14 in the memory 4 can also done from the outside by the circuit 11. The write circuit 14 must be driven with the designations of absolute addresses. As an example, the first instruction, INSTRUCTION 1, must be stored at address 0000 of the memory 4. In the state of the art, the circuit 14 acts under the control of the circuit 11 through the interface 12. The circuit 11 and those who control it must have the necessary knowledge and confidence to do this recording.

In the invention, to avoid untimely action of a circuit 11, particularly when it is desired to have

several types of uses in the area 4, no instruction is stored in the area 8 under the control of an external circuit 11.

To achieve this result, the operating system 5 of the invention is provided with a specific loading macro instruction which is symbolically represented here by a LOAD instruction from the operating system. The purpose of the LOAD instruction is to record in the memory 4 the instructions for an application program 15 and to reserve the necessary space in the memory for the data used by the program 15.

These two operations will now be described. The order in which they occur is not important as long as this is provided for in advance. Practically speaking, the program 15 and the definitions of the data used are presented to the interface 12 in the form of a file 17 which the LOAD instruction consults. In the invention, using a counter 16 which counts the instructions already recorded, an initial CPTI0 address is stored, showing that it is the first address of the instruction available for a new program.

The loading occurs in the following manner. Figure 2, the microprocessor 2, in executing the LOAD instruction will read (step 18) via the interface 12, the first instruction of the program 15, that of which the row is 1. The LOAD instruction then assigns (step 19), to this instruction of row 1 an absolute address equal to CPTI. At the beginning of the loading the CPTI

address is an initial CPTI0 address. Taking an arbitrary example, the CPTI0 address is 2356 in the memory 4. During the following step 20 of the LOAD instruction, this row 1 instruction is recorded in the memory 4 in the area 21 at the desired address: 2356. The assignment mentioned thus involves calculation of the absolute address. This assignment can include modification of the program to present the absolute address at the beginning of the instruction (or at the end) where it must be recorded. At the time of this recording, the LOAD instruction reads this absolute address and records the instruction at the corresponding address. With a test step 21, the LOAD instruction then verifies that the program instruction 15 (which has just been recorded) was the last to be recorded. As it is not the case here, at step 22 the contents of the counter 16 is incremented so that an area 23 of the counter 16 now includes a current increased value of CPTI (in an example, equal to 2357). Then the LOAD instruction triggers the return to step 18 in order to read a later instruction in the program 15, the row 2 instruction, and to record it at address 2357. This continues in the same way until all of the instructions are recorded.

A program such as program 15 usually includes jump instructions. As demonstrated in the program 15, a row *n* instruction will trigger a jump to a row *p* instruction. A jump instruction is usually a IF or GOTO

type instruction. The term branch instruction is also used, sometimes. In this case, row *p* is the destination of the row *n* instruction. For a program in a high-level language of a known type, the LOAD instruction includes a test 24 to check whether or not the row *n* instruction which is to be recorded is a jump instruction. This was not the case, so far. It is the case however for the row *n* instruction.

During step 25, the *p* destination (relative indication) is replaced by an absolute destination (an absolute address in memory). This absolute address is equal to the status of the counter of the initial CPTI0 instructions added to the value *p*. During the instruction 25, after having calculated the absolute destination address , the LOAD instruction modifies the row *n* instruction to transform it to an instruction in which the destination is no longer *p* (relative destination) but rather the new absolute destination. Then, at step 20 of the LOAD instruction, the row *n* instruction is recorded at address CPTI0+n-1 in area 21. The program goes on in this way until the recording of the *q* instructions of program 15.

If the sequence of rows 1, 2, 3, n, p, and q of the program 16 instructions have holes, there could be a LOAD instruction first to fill the holes (to calculate a continuous relative addressing), and then to modify a first time, in the event of a jump, the *p* destination addresses to take into account a continuous

sequence. In this case p would be replaced by p' (corresponding to the continuously changing rows) and step 25 of the LOAD instruction would apply to the value p'.

There are two possibilities for storing the data in the memory. The first, shown in figures 3a to 3c of the resul, index, temp, nain variables of the program 15 are detected during a first reading of the program 15. An address ordered @ data: 0 to @ data: 3, is gradually assigned to them. These relative ordered addresses are then transformed to absolute addresses (with the same mechanism as for the instructions). The program 15 is then modified to replace these relative ordered addresses by absolute addresses from 2450 to 2495. This is apparent in figures 3a to 3c.

Alternatively, the program 15 will have a DEFIN definition instruction with the quantity of space needed in memory as an associated argument. Here for example, the program 15 begins with the row 0 instruction, DEFIN combined with the argument 45. With this row 0 instruction, the LOAD instruction knows that it must reserve an area 26 in the data memory 4 to store the data used by the program 15. This makes it possible to use a variable length storage facility of the operating system 5 if necessary.

In both cases in the invention a second counter 27 is used to contain the start address available in the memory 4 to store the data and also a final address

taking into account the number of variables used in the program 15 or the argument here 45 indicated in the DEFIN order. The place reserved in the memory is intermediate between these two addresses. How counters 16 and 27 can be arranged differently will be explained hereunder. The explanation given here is intended to facilitate understanding of the invention's method.

In a preferred example, areas 21 and 26 of the memory are contiguous: one address follows the other. The starting address, containing the CPTI0 value of the counter (16 and the final address of the counter 27), determines the total occupation of the memory. In these conditions, the beginning of the counter of the data addresses 27), can be calculated by the LOAD instruction during an initial step 28). Thus, with a test analogous to test 22, it will be shown that the last instruction of row q was involved how many instructions there are is known and CPTD0 addresses and the following ones beginning with CPTI0+q are assigned to the data.

The flow chart in figure 2 is optional, it can be done differently as will be seen below. For example, if the program 15 has 94 instructions, q=94, the starting address of the CPTD0 data counter will be the address 2450 corresponding to the addition of the starting address of the CPTI0 counter and the number of instructions memorised. During step 29, the LOAD instruction then reads the row 0 instruction, if any,

and reserves the space of area 26 in memory, up to address 2495. Then modifications are brought, using with the LOAD instruction, at step 30, the program instructions to record the necessary absolute addresses: the jump addresses (address 2358 for the instruction of row 6 BNZE stored itself at address 2362) and the addresses in memory from the place where the values of the variables used in the program are stored.

Figures 3a to 3c show an explanation of the program for the purpose of its recording in the memory 4. Initially (figure 3b), the variables mentioned are replaced by relative ordered addresses for which the order corresponds to the order of appearance of the variables in the program.

In practice, the LOAD instruction transforms the program 15 shown in figure 3a into a program 15 shown in figure 3c. It might be wiser to reserve the data first, to assign to the data and variables the absolute addresses in the data memory 26 and then to assign their place to the instructions in the memory 4.

Once the program 15 has been loaded into areas 21 and 26 of the memory 4, the LOAD instruction records a descriptor at step 31 in an area 32 of the memory 4. The descriptor in area 32 is for the name of the application, in this case it would be the SERVICE application. It also relates to the starting absolute address 2356, and the arrival address 2495, where all

the program instructions and data are stored. The advantage of storing the instructions in area 21 at addresses of rows lower than those of the data of area 26 is that the program can start directly at address 2356. When it recognises the name of the SERVICE application, the descriptor in area 32 informs the operating system of the fact that the program involved starts at address 2356.

As indicated in program 15 and in figures 3a to 3c and according to the preferred mode of the invention, the instructions correspond to a high-level, symbolic language which itself refers to instructions which are executable by the microprocessor. For example, the purpose of MOVE is to copy the value which is given in the second argument to the address which is indicated in the first argument. The purpose of the MULT instruction is to multiply the contents stored at the address of the second argument of this instruction by the contents stored at the address of the first argument and to put all of this at the address of the first argument. Figure 4 summarises these aspects. It can be seen in particular that the BNZE instruction is the jump instruction.

Such a program is usually compiled before being recorded in the memory 4. The purpose of the compiling is to transform each MOVE, MULT, or other instruction – which are macro instructions – into sequences of micro instructions which can be directly executed by the

microprocessor 2. The purpose of these sequences of micro instructions is of course to accomplish the function of the macro instruction, but also to organise all of the readings in memory 3 and 4, all of the transfers to the bus 13, all useful releasing of register 7, etc.

To facilitate the work of the service providers, the invention does not require that they compile their program nor that they write the program 15 in machine language which could be executed by the processor 2.

Rather than storing the instructions in language which could be directly executed by the microprocessor 2, i.e. instructions which are loadable as is in the instruction register 6 of the microprocessor, it is preferred to ask the service providers to program their programs 15 in a high-level language (with instructions of the type found in figure 4.

In this case, the operating system recorded in the memory 3 is provided with a command interpreter 33 which is capable of transforming each of the macro instructions (MOVE, MULT, PLUS, etc.) into a sequence of micro instructions which can be directly executed by the microprocessor 2.

In the invention, the LOAD instruction is one of the instructions loaded into the operating system 5.

The command interpreter program 32 includes a sequence of micro instructions corresponding to this LOAD instruction.

The micro instructions are those mentioned by steps 18 to 31. When the application is implemented, an instruction (in high-level language) taken from area 21 sends a macro instruction to be interpreted by the interpreter program 33.

When the application is running, the application descriptor is used each time to verify that the addresses on which the program which is running acts, are addresses contained within the limits indicated by the descriptor in area 32.

The operating system and the interpreter program 33, when it is decided to implement one of these, should preferably be recorded in the non-programmable type memory 3 at the time of manufacturing of the integrated circuit 1. Thus, these parts of the program cannot be altered by the applications which are loaded, in the memory 4 for example.

Counters 16 and 27 have been described with registers. These counters could however be replaced by counting-type instructions executed by the microprocessor 2 and accompanying the LOAD instruction. The purpose of these counting instructions would be in particular to review the descriptors already present in the memory. The counters could also be more physical in appearance. The storing of the start values of counters 16 and 27 could be replaced by a reading of the last descriptors 32. When several applications are recorded in the memory 4, there are several descriptors. The

last one recorded is found by reading the memory backwards: this is the last location used before an empty location.

CLAIMS

1 - A method for loading an electronic chip (1) mounted on a chip support and including a microprocessor (2) and a memory (3,4), with an application program (15) for this chip support, this program containing instructions in which:

- the chip is provided with an operating system (5) including a memory loading program (LOAD), this operating system controlling the functioning of the microprocessor and the memory,

- the application program (15) is recorded (20) in the memory using the memory loading program

characterised in that

In the application program, a quantity of space needed in the chip memory (4) to contain the data to be used by this application program is defined (DEFIN, resul, temp, index...)

- during loading, this memory loading program calculates the absolute addresses to record the data used in the application program and the instructions for the application program in the memory,

- the application program is modified accordingly, and

- this loading program records the data and instructions present in the application program at the absolute addresses calculated.

2 - A method according to the claim, characterised in that we have the loading program records a descriptor (32) of the limits of the calculated absolute addresses at which these instructions and data are found.

3 - A method according to one of claims 1 or 2, characterised in that

- whether an instruction to be loaded is a jump instruction is checked and if this is the case,

- an absolute address for the jump destination, is calculated

- within this instruction, the jump destination is modified by replacing this destination by a calculated absolute jump address, etc.

- the modified jump instruction is recorded.

4 - A method according to one of claims 1 to 3, characterised in that

- the places in the memory are calculated as a function of the available space in this memory.

5 - A method according to one of claims 1 to 4, characterised in that

- the program data and instructions are loaded into continuous memory areas, and

- the limits of the addresses of these contiguous areas used for this purpose are recorded in the descriptor area of the memory.

6 - A method according to one of claims 1 to 5, characterised in that

- the chip is provided with an operating system which includes a command interpreter program (33),

- recorded in the memory of the chip, using the memory loading program is an application program (15) for the chip card including instructions written in a high-level language, which cannot be directly executed by the microprocessor, for the purpose of having the microprocessor execute the interpreted instructions which result from these instructions (which cannot be directly executed) once they have undergone interpretation by the command interpreter program,

- at the time of this recording, the memory loading program calculates the addresses to record the program instructions in a high-level language into the chip memory,

- the high-level language instructions based on this calculation are modified,

- these high-level language instructions are recorded in the memory at absolute addresses calculated by the loading program.

7 - A method according to claim 6, characterised in that

- the application program is executed, verifying with each interpreted instruction to be executed that the data or instructions to which an interpreted instruction refers are found at an address contained within the limits recorded in the descriptor of the decision table.

8 - A method according to one of claims 6 to 7, characterised in that

- recorded in the chip memory is the operating system which includes the memory loading program and the command interpreter program from the time this chip is made.