(12) UK Patent Application (19) GB (11) 2 284 492 (13) A

(43) Date of A Publication 07.06.1995

(21) Application No 9325002.5

(22) Date of Filing 06.12.1993

(71) Applicant(s)
Graeme Roy Smith
4 Bridgemere Close, Radcliffe, MANCHESTER, M26 4FS, United Kingdom

(72) Inventor(s)
Graeme Roy Smith

(74) Agent and/or Address for Service
Graeme Roy Smith
4 Bridgemere Close, Radcliffe, MANCHESTER, M26 4FS, United Kingdom

(51) INT CL^6
G06F 9/30

(52) UK CL (Edition N)
G4A AJR APX

(56) Documents Cited
GB 1529538 A **US 5179680 A**

(58) Field of Search
UK CL (Edition M) **G4A AJR APL APX**
INT CL^5 **G06F 9/30 9/44**
On-line database: WPI

(54) Data processor with instruction decompression unit

(57) A computer control unit used for instruction sequencing in microprocessors, microcontrollers and digital signal processors has or is used with an instruction decompression unit 5. The unit 5 decompresses a previously compressed instruction block and provides sequential instructions to an instruction decode unit 7 at the correct time. Decoded instructions are passed in turn to instruction execution unit 8. A multiplexer 6 allows the use of non-compressed instructions, by bypassing unit 5.

Method and apparatus are provided for source code compilation, de-bug, code simulation and code compression for use with a hardware instruction decompression unit 5. A compressed program is loaded in to the system main memory. Required and potentially required instruction blocks are loaded in to an instruction cache 3 by the computer control unit at run time.

Program compression allows for reduced memory requirements and greater data and instruction through-put, by reducing main memory accesses.

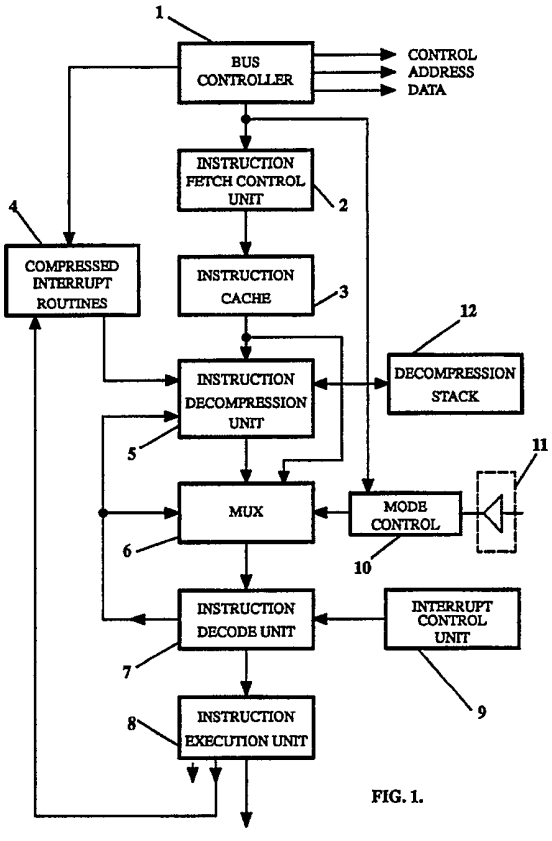

FIG. 1.

GB 2 284 492 A

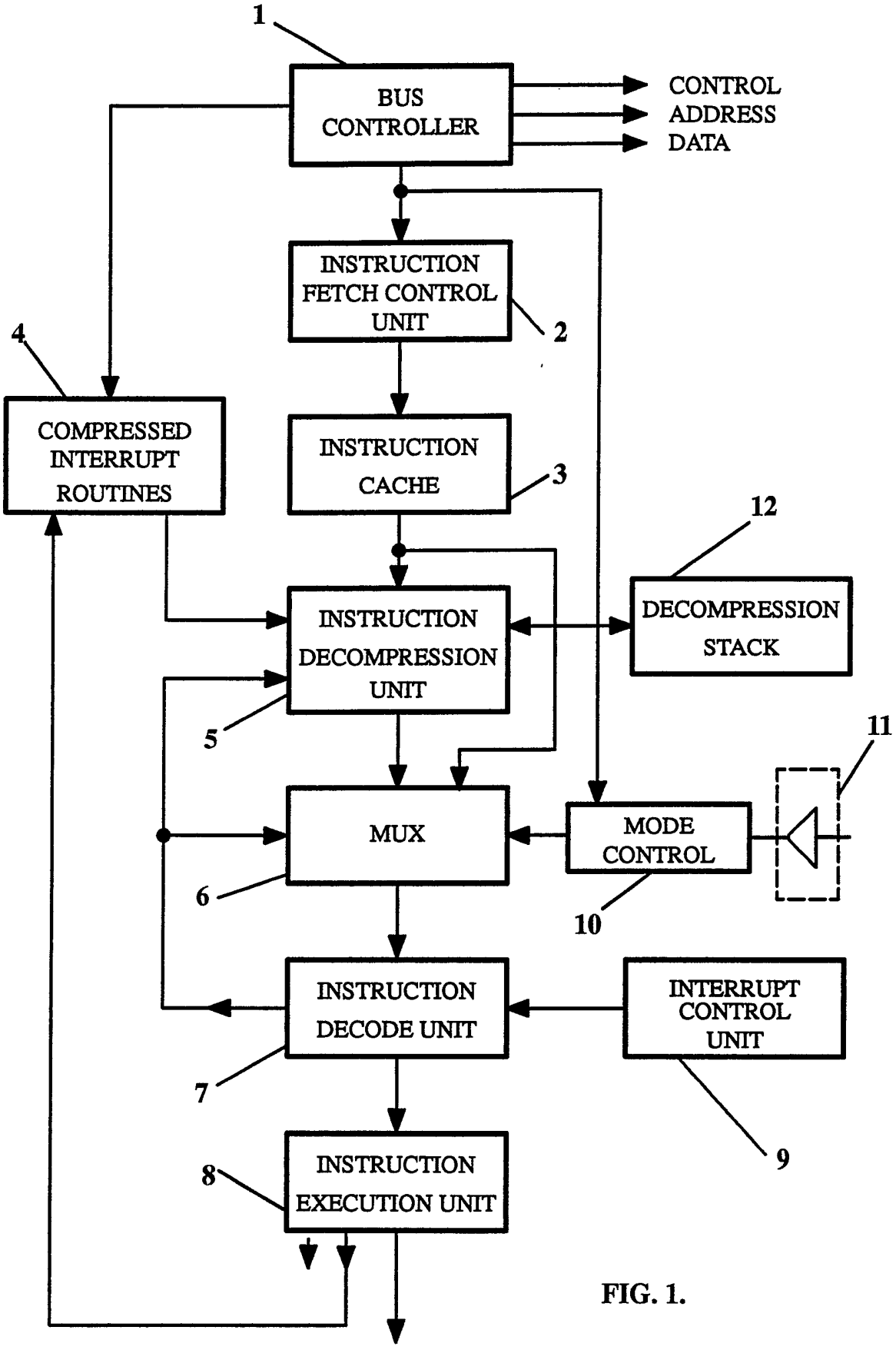
1
BUS CONTROLLER
CONTROL
ADDRESS
DATA
INSTRUCTION FETCH CONTROL UNIT
2
4
COMPRESSED INTERRUPT ROUTINES
INSTRUCTION CACHE
3
12
INSTRUCTION DECOMPRESSION UNIT
DECOMPRESSION STACK
5
11
MUX
MODE CONTROL
6
10
INSTRUCTION DECODE UNIT
INTERRUPT CONTROL UNIT
7
9
8
INSTRUCTION EXECUTION UNIT

FIG. 1.

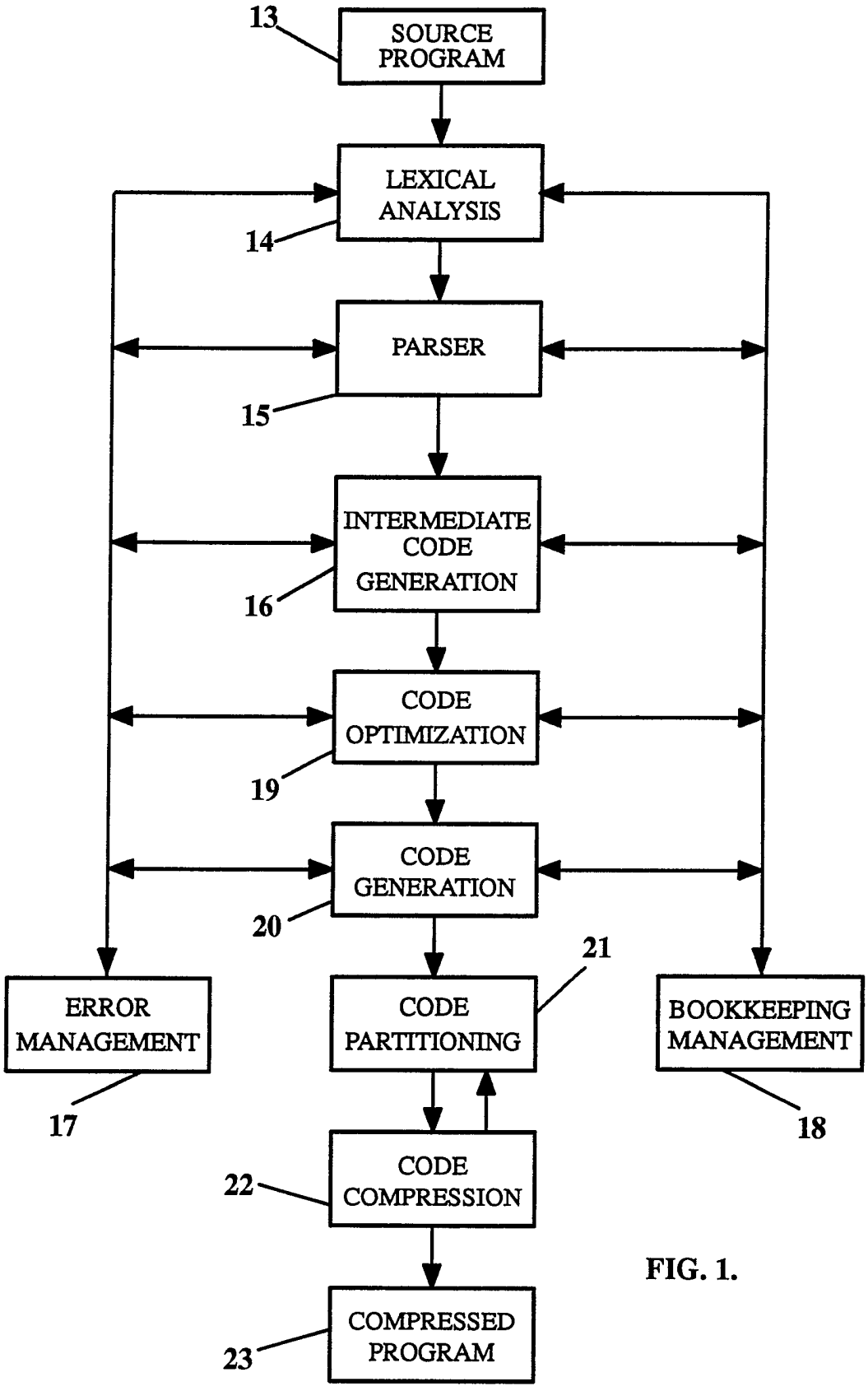
13
SOURCE PROGRAM
LEXICAL ANALYSIS
14
PARSER
15
INTERMEDIATE CODE GENERATION
16
CODE OPTIMIZATION
19
CODE GENERATION
20
21
ERROR MANAGEMENT
CODE PARTITIONING
BOOKKEEPING MANAGEMENT
17
18
CODE COMPRESSION
22
COMPRESSED PROGRAM
23

FIG. 1.

2284492

IMPROVEMENTS TO COMPUTER CONTROL UNITS

This invention relates to a computer control unit and source code compiler.

There are several forms of VLSI (Very Large Scale Integration) silicon computer devices available today, namely CISC (Complex Instruction Set Computers) and RISC (Reduced Instruction Set Computers) microprocessors, microcontrollers, and digital signal processors or DSP's. These integrated circuit devices are used to execute various instructions which manipulate binary data, arithmetically and logically in an arithmetic logic unit (ALU).

Each manufacturers computing device has its own dedicated instruction set. To perform any useful function, individual instructions are concatenated in a structured and coherent way to form a program. This program is stored in main memory. To execute a program, the computer loads these instructions one at a time into its instruction register, either directly or via an instruction cache. The instructions are then decoded and executed accordingly. The instruction cache allows groups of sequential instructions to be stored locally so the instruction execution unit can access individual instructions quickly without having to perform time consuming external memory read operations. This mechanism therefore increases system performance by reducing main memory accesses and allows instructions to be executed at a faster rate.

Many programs are thousands of instructions long and require large memory arrays to store them. These requirements can range from several kilo-bytes to many mega-bytes of memory. The instruction caching requirements also become more complex and involve sophisticated cache hit, cache miss and cache/main memory consistency control mechanisms. This includes cache write-through, write-back and snoop mechanisms. The instruction cache stores groups or lines of instructions that are regularly used or have a high chance of being required by the instruction execution unit. By making more of the program software directly available to the computer control unit and reducing main memory accesses, through the use of software compression algorithms and hardware decompression techniques, program execution rates can be increased and the size of the memory required to store the programs can be reduced.

With today's high gate densities, it is possible to fabricate three to four million transistors on a single silicon die. These densities allow for increased device functionality, system performance, data and instruction through-put and can be used to integrate hardware instruction decompression units either on the same silicon die or as a separate co-processor.

According to the present invention there is a computer control unit comprising a bus controller, an instruction fetch control unit, an instruction cache to store likely required groups of instructions, an instruction decompression unit which decompresses a block of compressed data from the instruction cache, an instruction decode unit together with a multiplexer to select the instruction source, an instruction execution unit, an

interrupt control unit and an compressed interrupt routine memory which stores compressed interrupt service routines.

Apparatus and method for optimizing the source program into compressed target code are also provided. A compiler will first be used to check the source code (program) for correctness. Error free code will then be compressed by the compiler before being loaded into main memory for execution by the host processor. The compression algorithm can be a lossless compression technique or compression algorithm that allows some loss. In the latter case, the instruction decompression unit can regenerate the instruction codes from the 'lossy' compression block due to code redundancy, in a similar fashion to that performed by error correcting codes.

Due to program compression the program memory requirements are reduced because the program can be stored in a significantly smaller memory. As blocks of the program are compressed, more sections of the program can be loaded in to the computer control unit's cache memory. This increases the chance of a cache hit and hence processor performance. At the correct time, appropriate compressed instruction blocks are loaded in to the instruction decompression unit. The instruction decompression unit implements a decompression algorithm which decompresses the compressed instruction block. Sequential instructions are then generated at the output of the instruction decompression unit which are passed to the instruction decoder.

Compressed interrupt routines are stored in the local compressed interrupt routine memory. When an interrupt occurs instruction execution is halted at an appropriate time and the corresponding compressed interrupt routine loaded in to the instruction decompression unit for execution. Upon completion of the interrupt routine the computer control unit returns to normal program execution, unless there is another interrupt of less priority waiting to be serviced. Program flow integrity and decompression parameter maintenance is achieved via a stack associated with the instruction decompression unit which stores and retrieves parameters used by the instruction decompression unit as necessary.

A specific embodiment of the invention will now be described by the way of example with reference to the accompanying drawings in which:-

Drawing 1/2 figure 1 shows a block diagram of the computer control unit implementing an instruction decompression unit;

Drawing 2/2 figure 1 illustrates the program compilation process and target program compression by use of a flow diagram.

Referring to drawing figure 1 the computer control unit comprises a bus controller 1 which is used to gain access of the local bus, a memory address register to hold the address of the memory location to be accessed and a bi-directional data bus on which data is read from or written to main memory or external cache memory. The appropriate control signals are also provided to access peripheral functions.

As a program is executed the instruction fetch control unit 2 loads blocks of program data into the instruction cache 3. This program data can be compressed data or uncompressed data. The mode of operation is dictated by a mode bit in a control register 10, an I/O pin 11 of the device or the output of the instruction decode unit 7. This allows the computer control unit to run either uncompressed programs in the conventional sense, a compressed programs 23 which have been previously compressed by an appropriate compression algorithm 21, 22, or a program which is a mixture of both compressed and non-compressed code. If the program data is not compressed then the instruction decompression unit 5 is bypassed and instructions from the instruction cache memory 3 are input to the instruction decode unit 7 via the instruction source multiplexer 6. The instructions are decoded and transferred to the instruction execution unit 8 where appropriate actions are taken.

In program compression mode, the source program must first have been compiled, checked and edited for correctness and then compressed as outlined in the flow diagram shown in figure 2. Target program code generated by the code generation function 20 of the compiler is passed to the code partitioning function 21 which divides the program into manageable sections of code. This partitioning phase is dependent on the compression algorithm implemented by the code compression 22 section of the compiler 14, 15, 16, 17, 18, 19, 20, the code size and the occurrence of Branch/Call/Goto or Jump type instructions. The decision to perform a branch in the instruction flow is taken by the instruction execution unit 8. If the instruction flow of the program dictates that a branch occurs to an area of code not contained by the current instruction block in the instruction decompression unit 5, then the appropriate instruction block will have to be fetched from the instruction cache 3 or main memory and loaded in to the instruction decompression unit 5. A successful branch to an area of code within the current compressed instruction block, contained in the instruction decompression unit 5, will cause the instruction decompression unit 5 to be initialised with parameters that will allow the instruction decompression unit 5 to generate instructions and permit a continuation in the program flow. The location of code pointed to by a branch type instruction is identified and marked by the code partitioning function 21. Code implemented at these potential branch destinations is coded in such a way by the code compression function 22 that the instruction decompression unit 5 is able to initialise the instruction decompression unit 5 and allow the continuation of program flow.

Compressed instruction blocks 23 are fetched from main memory by the instruction fetch control unit 2 via the bus controller 1 and stored in the instruction cache 3. If the required instruction block is stored in the instruction cache 3 then this block of data is pre-loaded in to instruction decompression unit 5. The instruction block can be pre-fetched and initialised in the instruction decompression unit 5 before the instructions are required to be decoded. This is due to the multi-stage pipelining provided in the instruction execution unit 8. If the required instruction block is not available in the instruction cache 3, that is, there is a cache miss, then the instruction block is fetched

directly from main memory by the instruction fetch unit 2 and loaded into both the instruction cache 3 and the instruction decompression unit 5.

When activated by the decompression mode control bit 10, the I/O pin 11 or the output from the instruction decode unit 7, the instruction decompression unit 5 will decompress the loaded instruction data block and generate sequential instruction op-codes. These op-codes are passed to the instruction decode unit 7 via the instruction source multiplexer 6. By being able to select between the output of the instruction decompression unit 5 and the instruction cache 3, the computer control unit can execute instructions that are compressed or non-compressed. This scope allows for mixed mode programs 23 where a program 23 is constructed from both compressed instruction blocks and non-compressed instruction blocks. The output from the instruction decode unit 7 is transferred to the instruction execution unit 8 at the appropriate time. Output control fields from the instruction execution unit 8 are used to control peripheral logic which performs the desired function.

Compressed interrupt routines are stored in the local compressed interrupt routine memory 4. When an interrupt occurs instruction execution is halted at an appropriate time and the corresponding compressed interrupt routine loaded in to the instruction decompression unit 5 for execution. Upon completion of the interrupt routine the computer control unit returns to normal program execution, unless there is another interrupt of less priority waiting to be serviced. Program flow integrity and decompression parameter maintenance is achieved via a decompression stack 12 associated with the instruction decompression unit 5 which stores and retrieves parameters used by the instruction decompression unit 5 as necessary.

CLAIMS

1 A computer control unit comprising an instruction decompression unit for decompressing program instructions previously compressed on a compiler suitable for compressing the program source code in to compressed instruction blocks, the instruction decompression unit implements a decompression algorithm, this algorithm performs the inverse function of either a lossless compression algorithm or a compression algorithm that allows limited code loss, the original code being regenerated by the instruction decompression unit due to redundancy in the instruction codes used by the computer control unit, means for fetching the required compressed instruction blocks from main memory or cache memory to an instruction cache by the computer control unit, a means for transferring the compressed instruction blocks from the instruction cache to the instruction decompression unit, so a sequence of individual instructions can be generated from the compressed code for input in to the instruction decode unit via an instruction source multiplexer which can select between decompressed instructions from the instruction decompression unit or non-compressed instructions from the instruction cache, thus allowing the computer control unit to run either compressed programs, non-compressed programs or a program that contains both compressed and non-compressed instructions, the control of the instruction source multiplexer being determined by an input pin to the computer control unit, a code pattern written to a mode register or the output from the instruction execution unit, the decompressed instructions being transferred to the instruction decode unit at the correct time, before being passed to the instruction execution unit which performs the intended function, associated with the instruction decompression unit is the decompression stack for storing intermediate decompression parameters if interrupts occur and ensuring instruction decompression unit parameter maintenance and integrity, these parameters being restored into the instruction decompression unit in reverse order to enable continuation of program flow after an interrupt has been serviced, unless another lower priority interrupt is pending.

2 An instruction decompression unit which is separate to the main computer control unit and is employed as a computer control unit co-processor for the sole purpose of performing instruction decompression and transferring the individual decompressed instructions to the instruction decode unit in the computer control unit.

3 A computer control unit substantially as described herein with reference to Figures 1-2 of the accompanying drawing.

Patents Act 1977
Examiner's report to the Comptroller under Section 17
(The Search report)

Application number
GB 9325002.5

Relevant Technical Fields

(i) UK Cl (Ed.M) G4A (AJR, APL, APX)

(ii) Int Cl (Ed.5) G06F (9/30, 9/44)

Search Examiner
B G WESTERN

Date of completion of Search
3 FEBRUARY 1994

Databases (see below)
(i) UK Patent Office collections of GB, EP, WO and US patent specifications.

(ii) ONLINE DATABASES: WPI

Documents considered relevant following a search in respect of Claims :-
1-3

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family; corresponding document.

Category	Identity of document and relevant passages		Relevant to claim(s)
X	GB 1529538 A	(IBM) see whole document, but especially page 5 lines 52-55	2
X	US 5179680 A	(COLWELL ET AL) N B columns 15-21	2

Databases: The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journal (Patents). The on-line databases considered for search are also listed periodically in the Official Journal (Patents).