1 599 421

PATENT SPECIFICATION (11) 1 599 421

(21) Application No. 18616/77 (22) Filed 4 May 1977
(23) Complete Specification filed 11 April 1978
(44) Complete Specification published 30 Sept. 1981
(51) INT CL³ G06F 11/00
(52) Index at acceptance
G4A 12N 13E AP

(72) Inventor KENNETH JAMES HAMER-HODGES

(19)

(54) IMPROVEMENTS IN OR RELATING TO INFORMATION PROTECTION ARRANGEMENTS IN DATA PROCESSING SYSTEMS

(71) We, THE PLESSEY COMPANY LIMITED, a British Company of Vicarage Lane, Ilford, Essex IG1 4AQ, do hereby declare the invention, for which we pray that a patent may be granted to us, and the method by which it is to be performed, to be particularly described in and by the following statement:—

The present invention relates to data processing systems and is more particularly concerned with such systems incorporating information protection arrangements employing so-called "capability" addressing techniques.

Such addressing techniques are extensively disclosed in Patent Specification Serial Nos. 1,329,721; 1,329,722; 1,344,474 and 1,410,631. Capability addressing structures are based upon the fundamental of allocating an absolute and permanent name and access key to each system resource. All resources or logical units are allocated a capability regardless of whether they are storage segments (held in main or backing store) or peripheral equipments. Each logical unit is defined by a unique capability and can be accessed only by programs (or processes) which possess a copy of the correct capability. A list, called a system (or master) capability table, is incorporated in a storage segment having an entry for each accessible resource and each process is provided with at least one list (called a reserved segment pointer table) of capability pointers which are relative to the system capability table entries. The capability pointer list, therefore, defines the resources to be used by the process. Each capability pointer also includes the access code information which indicates the resource type and the permitted mode of access to the resource. It should be noted that the set of capabilities owned by a process is itself a logical unit and is consequently defined by a further capability. Typically, the access rights may be used to define (i) execute rights for program segments (ii) read-only rights for data segments, (iii) read/write access rights for data segments, (iv) read only or read/write rights for capability segments and (v) enter access rights. The enter access right defines capabilities relative to sub-routines and hence enter capabilities are accessed by program call instructions which transfer control to a subroutine program defined by the capability held in the enter segment.

It is an aim of the present invention to expand the flexibility of the capability pointers of a capability addressing system of the above mentioned type.

According to the invention there is provided a data processing system incorporating capability addressing arrangements of the above mentioned type in which the access code resource type indicator in a capability pointer defines (i) an inform resource (ii) an outform resource and (iii) a passive resource and wherein when a capability pointer relative to a passive resource is detected during a load capability register operation that register is conditioned such that an automatic trap operation is performed when it is use causing entry into a trap handling process through a passive resource table the descriptor for which is held in one of the hidden capability registers and the information held in the passive capability pointer is used to define the passive resource table entry.

Typically, passive resources involve system control operations such as the allocation/adjustment of capability pointer access codes, movement of objects in and out of main store, the transformation or deletion of capability pointers or the association objects and the access to values not defined by capabilities for example. In addition the handling of the creation of SCT entries and the associated sum checks can be included within the sphere of passive resources. Further access operations involving internal registers and test

operations can all be achieved using passive resource accessing techniques.

In data processing systems using capability addressing it is also necessary to perform administrative tasks on the information segments to control the administration of the main store. Each information segment has external control fields that must be addressed by the "store manager" process which has therefore to be given a wide access capability covering the whole of the main store. This technique requires that the store manager be fully trust worthy and must not be capable of being invaded since a privileged mode has been created permitting overlapped capabilities. Such circumstances lead to the fear that the privacy of a data base may be compromised. Accordingly it is a further aim of the invention to provide a modified "inform" resource which overcomes this short coming.

According to an embodiment of the invention there is provided a data processing system of the type incorporating capability addressing arrangements and having an inform segment comprising a segment header and a user object segment and normal inform references involve access to the user object segment only whereas the processing system includes means activated when an operating system process is being executed to confine manipulative access to the segment header of a segment or to permit read-only-access on a serial transfer basis to the complete inform segment.

In capability addressing systems of the type disclosed in Patent Specification Serial No. 1,329,721 the size of the system capability table (SCT) is important since it is an overhead on the system which cannot be used for the storage of application dependent data. Accordingly it is another aim of the invention to provide arrangements which reduce the dynamic requirements for the SCT.

According to the embodiment of the invention there is also provided a data processing system of the type incorporating capability addressing arrangements using a system capability table and reserved segment pointer tables and in which each inform capability pointer includes a system capability table sequence number and a pointer value and a plurality of system capability table sections are provided each section being identified by a discrete sequence number and the pointer value is arranged to define an entry in the table section identified by the accompanying sequence number.

Advantageously the table sections may be distributed throughout storage modules which make-up the main store.

The sequence number which make-up the main

The sequence number can be used to identify a primary SCT section when say equal to zero or an alternative secondary SCT section when the sequence number is not equal to zero. The sequence number in the pointer must match a sequence number held in the SCT entry addressed for a successful load capability register operation to take place. In such an arrangement the primary SCT section would satisfy all store resident blocks, application code and peripherals whereas the secondary SCT section (s) could be used to store overlay and dynamic blocks thereby simplifying the SCT slot allocation and garbage collection process.

Finally in data processing systems using capability addressing arrangements sub-routine call operations are performed using capability pointers which include a so-called "enter" access right code which is used to validate the pointer used on a sub-routine call operation. The pointer used on a sub-routine call operation defines an enter capability segment for the sub-routine itself. On many occasions a common sub-routine is accessible by a number of calling processes however some or all of these processes may have differing access rights necessitating duplication of enter capability segments. It is yet a further aim of the present invention to overcome the duplication of enter capability segments in such circumstances.

The embodiment of the invention also provides a data processing system using capability addressing arrangements in which each enter capability pointer includes a plurality of enter access bits and each enter access bit when in a first state is indicative of the right to use a given linear offset of the enter capability segment whereas when an enter access bit is in a second state it is indicative that the calling process does not have the right to use the so defined offset.

The invention will be more readily understood from the following description of one embodiment which should be read in conjuction with the accompanying drawings. Of the drawings:—

Fig. 1 shows in block diagram form the central processing unit incorporating equipment suitable for use with the features of the invention.

Fig. 2 shows the configuration of various capability pointers for use with the features of the invention,

Fig. 3 shows the items involved in a capability transaction using a sealed object segment,

Fig. 4 shows the vectored call structure for passive resources,

Fig. 5 shows a table of capability indicators,

Fig. 6 shows the differing access maps to the same resource using a re-coded enter access field,

Fig. 7 shows the sequence number reference to a segregated system capability table,

Fig. 8 shows an implied capability,

Fig. 9 shows the capability compare instruction,

Fig. 10 shows the operation of the mask access function,

Fig. 11 shows a modification to the flow diagram for the load capability register instruction to incorporate the sequence number feature,

Fig. 12 shows an enhanced load capability register instruction flow diagram,

Fig. 13 shows a table of interrupt actions while,

Fig. 14 shows the flow diagram of an enhanced "trap on use" sequence.

A capability as mentioned above is "an unforgettable name providing access rights to a specific object in memory". This concept provides the basis for secure access to objects (capability segments) from a range of subjects (Enter resources) through a protection matrix (the system Capability Table).

In the system disclosed in the above mentioned Patent Specifications the protection matrix is the combination of the System Capability Table (the object list) against capability pointer lists (reserved segment pointer tables) available to various subjects. The most primitive subject would be an Enter domain. Further the matrix is dynamic through the use of secure Store Capability (and Load Capability) instructions.

In addition only a subset of the full matrix need be present in fast store at any time. The full matrix is present only through the mechanisms of backing store management (explicit or virtual memory).

Recognition and Response to Alternative Capability Forms

The current inform capability pointer construction is formed from three fields; the offset (16 bits) the access (6 bits) and the control (2 bits). The pointer is the basis of all access control and therefore critical to system power.

For improved efficiency and greater flexibility the control field may be used in order to identify alternative capability forms as shown in Fig. 2. The control Field CT is arranged to have four states corresponding to the four principal interpretations of a capability pointer; (i) that the segment exists in store (inform), (ii) that the segment is not resolved into SCT (outform), (iii) that the word is a 22 bit data value or (IV) that the capability is passive in type (not a segment). For improved performance the microprogram hardware μPROG is arranged to respond uniquely to each control type through the code detector CD (Fig. 1) and the control type signals CTS. (At present only informs are responded to instantaneously and others are trapped on-use). Instead the outform could trap "on loading" (not "on-use" as at present) this would minimise the number of inform/outform conversions required to those used, rather than those present in store.

The data value pointer conditions the μPROG so that a fault entry is made "on-use" and passive pointers are interpreted by μPROG to access a passive instruction micro-sequence or to trap "on-use" to a new dumpstack subroutine level through a passive type table (held in one of the hidden capability registers for example CR15).

A New Definition for Inform Capability

The basic structure of the system memory will acknowledge new primitive types; Sealed Objects, Headers and SCT entires (in addition to Data, Capabilities, Programs and Enter Resources) see Fig. 3.

A sealed object is the combination of the user object and its co-located header. The only operation that can be performed on a sealed object is an unbreakable MOVE, which transports the object from locations (a to a+n) where n is the size of the sealed object. Other random access attampts are prevented by hardware checks.

Segment control parameters are in two parts, first a set of m words situated (at the base of a memory frame) immediately preceding the conventional (or user) object base address and second the descriptors (the SCT entry). Operations (new secured instructions) must be provided in order to permit the system to access the control parameters of any sealed object without the present need for wide access capabilities. The secured instruction should manipulate the "null" access (safe pointer) into a base/limit register permitting access to the selected control words (SCT or block header) as required through the combined use of a Call to a passive capability instruction.

A segment header may typically be used to hold information indicative of the identity and location of the corresponding segment on the disc backing store. Further information relative to segment type (e.g. program, data, capability pointer table, read-only data) and modification operations performed if any may be held in the header for use in validating the loading of a

capability register and in the operating of the store manager process respectively.

A Proposal for Passive Capabilities

Passive Capabilities provide a new dimension in access control. Since the interpretation of a passive pointer is not directly related to a unique store segment(s) the implementation is less store consuming and in certain situations therefore most efficient. However a mechanism is required to interpret passive capabilites such as the Trapped call "on-use" referred to earlier.

A Passive Capability is indicated by the two bit control field CT setting equal to value 1 say as shown in Fig. 2. This provides an escape code from the Inform or Outform format contraints and permits a new set of field boundaries within the capability word.

Apart from the two bit control field CT at least two other fields exist. The principal field would be a Type indicator field ATC to select or identify the specific function to be performed. The second field would be format dependent upon the type, as such it can be referred to as the Argument field, which may be made up from a number of smaller items.

Sufficient types can be provided to satisfy long term unknown demand over and above those identified at present. Perhaps a six bit ATC type field providing 64 directly accessible type classes would be a suitable minimum. The machine hardware would respond to certain type codes directly and trap to an available type handler for others through a trap table. Unused type fields would fail through an invalid type handler.

Inform Capabilities

Even if the steps are not undertaken to extend the types of capability handled by the system to include passive capability pointers there are a number of useful modifications concerning just the principal type of Inform pointer that exists at present. These modifications concern Access rights and Segment Identity.

The Access Field

Considering Inform and Outform pointers only, the access field is more difficult to consolidate than the other field. This is because the access type code uses linear coding for fault detection purposes. In practice less than 16 variations are used out of 64 possibles. However capabilities are internal representations and in that sense they do not need to be compatible between equipment generations. Therefore apart from the restrictions on mixed equipment solutions and human preferences it would be quite feasible to restructure the access field. This could take the form of coded access rights rather than a bit significant accesses rights thereby releasing a number of bits for other use.

An alternative enhancement alters the significance of the Enter access in order to enhance access control into shared subroutines. The principal followed is that access right control of all software levels should be consistent. Therefore, the permitted access to an Enter resource should in general be indicated by the Access Field (not a block of Fan out code). If this rule is followed then the inefficiency of multiple Enter blocks for each access variation and the overhead of fan out code is removed. In such a system the Enter access bit becomes a high level access right (an escape code from Hardware significant concepts e.g. Read—Write, Capability—Data—Program). When used "Enter Capabilities" define abstract resources which in themselves have abstract (and variable) access rights, e.g. Post Access or Wait Access on a Flag are different access rights, similar in concept to Write and Read but at a higher level see Fig. 6. In the same way as different users are offered different (Read or Write) access at the low level they should be offered at higher (Enter) levels and controlled through hardware validation. Each access bit EAB associated with an Enter block would therefore indicate the right to call a given linear offset of the enter block.

The Offset Field

It is expensive to create an SCT block greater than 10% of total fast storage. In general most fast storage (say 120K) will be core resident with an SCT demand of up to 10 K words (2K blocks of 60 words/block and 1K outform SCT entries). Beyond the core resident segments exist the virtual memory (VM) segments which are overlaid into any free areas. For the VM segments the pointers may be absolute when an SCT slot always exists as with or SCT may be dynamically assigned. One of the problems in the dynamic assignment case is the small size of SCT (10K or so). One solution is to accept a much larger SCT but even so it is unlikely, due to cost of memory, that one of 64K words would be constructed. This leads directly to the conclusion that the direct use of a 16 bit point offset is unnecessary. A more reasonable size would be a 13 bit offset (relating to experience with ROS/PDOS etc). The remaining bits can now be used for a number of alternative functions the most useful being a sequence number SN and or the identification of alternative SCT registers to ease store contention as indicated in Fig. 7. The three bit sequence number SN can be used to identify either a primary SCT when the

sequence number is equal to zero or an alternative secondary SCT when the sequence is not equal to zero. The sequence number in the pointer and the SCT limit entry sequence no code must match for a successful load capability register to take place. This operation is performed in Figs. 1a and 1b by comparing the SN values as the pointer and limit word pass through OPREG using CD.

The primary SCT would satisfy all store resident blocks in particular binary block B and the application code. The secondary SCT would be used for store overlay and dynamic blocks since the sequence number mechanism make garbage collections and SCT allocation easier. This would be further eased if the load sequence trapped when a disparity of sequence numbers was encountered.

Passive Capability Types

The advantages of a Passive Capability type as mentioned above arise from the ability to use the pointer as a fixed value not related to the SCT directly but translated indirectly through a subset trap table. If six bits of the passive capability pointer are used for ATC there remains a large field of 16 bits that are available for use by the type handler for alternative use.

Consider the well known Flag structure defined by Dykstra. Each flag is allocated a specific Enter Capability occupying memory space in the form of SCT entries, headers and segments. Although all common facilities are shared between all flags the overheads are substantial.

If the flag structure is implemented by a Passive Capability technique then only a single set of SCT, header and segments would be required positioned in the hidden trap table, and each independent Flag would be identified by the constant (secure) value held in the argument field AF (Fig. 4) of the capability pointer.

A whole range of complex resource types can be effectively implemented in this way. Additionally a complete range of privileged internal instructions as previously mentioned for system control purposes can be included in the form of directly executable passive or internal instructions.

Three particular example types of passive capability pointers are suggested as an introduction to the useful nature of this alternative.

Instruction Types

In order to control the overlay of the full protection matrix into the fast store subset, each object has associated with it control parameters. These parameters are not recognised as internal of the object since this could cause misaddressing and corruption. Thus the control is applied externally and is not addressed in direct association with the assigned capability or object.

At any instant there are a large number of objects in fast store. Each object has 'external' control fields (located in the SCT and a two or three word block header) that must be addressable by the "Store Manager". To do so via conventional capability addressing would require three times the number of capability blocks present at any time and would not provide a general solution since more heads and SCT entries would be generated. Therefore a tolerable solution is to provide wide access capability covering the whole of store for the store manager and trust in error free operations with regard to SCT and header access. Even ignoring hardware errors this solution presents two principal problems.

Firstly the store manager is a complex software operation that could not be expected to perform error free, or be "proof" tested. In effect a classic case for capability controlled operations. Second a "privileged" mode has been created which permits overlapped capabilities, this would destroy the confidence one could place in a totally private data base.

Instead the "Store Manager" should act as a secretary, fetching and storing information from the backing store to the main store. A secretary is given only selected rights to any object, those of "movement", "location" and in some cases "cleansing". Any object could then only be placed into memory by the "secretary" and not accessed on a random access basis.

In principal two "secretarial" modes of primitive access can be envisaged; MOVE access and CONTROL access. These modes of access would provide primitive secure facilities for the store manager to control the overlay of objects into fast store without the disadvantage of the present system.

"Operating system" access would be confined to secretarial modes of access only, derived from a "null" access field inform pointer (user provided) and a protected Passive Capability Instruction which would take a null access inform pointer NF detected by CD and provide the selected capability register loaded with the bounds set to the (BASE—3) and a special MOVE access right (26th bit of Limit Register) set by signal μM.

The only operation that could be performed when using this capability register is an INDIVISABLE MOVE instruction between store blocks (address bit 23=0) and peripheral controllers (address bit 23=1) monitored by the

comparator COMP and the microprogram μPROG.

The indivisable move cannot make Random Access to any segment but only serial transfer access in total as a sealed segment transfer.

Other Passive instructions might include Create Capability, Load SCT Slot, Load Header Slot or Increase Access. Additionally the Internal Mode capability could be eliminated and replaced by a set of passive instructions such as Set Interrupt Inhibit, Read Historic Register, Load Second Eight Capability etc. A full set of such instructions would therefore permit much finer control over the critical system control functions and in the limit a system could be defined that provided secure facilities to offer complete privacy and security over all user segments and functions.

For example if the Create Capability Instruction always operated by dividing down or splitting an existing capability a given sub-system could perform store allocation independently from the Operating System and other sub-systems. Therefore the only sub-system *ever* to possess a random access right capability to this portion of store would be the controlling sub-system thus ensuring privacy of access.

Implied Capabilities (Fig. 8)

Consider the case concerning "private" (transactional) data for any re-entrant routines of procedures. The present system demands that if a program is reentered then all transactional data and or capability values must be process based. Therefore they are rooted into the addressable portion of the process Dumpstack (D0—D7 or C0—C7). If any domain of operation requires transaction based storage in excess of the register set then this must be rooted in a capability register. There are a number of alternatives, one solution creates capability register C5 by convention as the root of all process based storage. Unfortunately this permits all domains to access the process based data of any other domain in the process. In consequence C5 is restricted in use, open to abuse by corrupt subroutines and course in its control.

This problem can be cured by the use of "implied" or indirected capabilities. In such a solution the addressable portion of Process Dump Stack can be extended beyond the machine registers to memory locations. Equally as important these locations can be unavailable to all but one subroutine level through the conventional capability protection.

Thus the system would retain the vertical transactional based isolation and add a horizontal domain based isolation with the benefit of removing the conventions surrounding C5 and freeing it for general use. Note that current software need not take the advantage of this facility in order to run on the improved machine, but if used it would vastly simplify the register save functions of the operating system.

The implied address held in the form of a passive pointer would be translated on use not on loading (c.f. conventional indirect addressing) thus it would be possible to store both the implied address by using a store capability instruction; or a dynamic pointer into the implied address by using the implied address as the operand addresses through the C field of the instruction.

The indirection should be indicated from the capability pointer. A possible implementation is shown in Fig. 8. Note that the implied capability register can be one of 16 which permits the indirect addressing of any hidden capability register. Care must be taken when referencing the dumpstack because it is a mixed block, a solution would be to check that the indirect offset is greater than the pushdown pointer size and that the IAR, C6 and C7 entries are nulled on any return instruction.

Accounting Type

One of the areas not effectively handled within the system at present is accounting. A simple proposal which would provide for more flexibility than usually possible on conventional systems would employ a passive capability type to perform the accounting functions.

The first form would use the argument field from the pointer to identify an accounting cell which would be adjusted according to the accounting rules for example the size of store in use. Limits would apply and a user would be restricted and billed accordingly.

The second form would be effectively prepaid and the argument would itself be the account cell. Once reduced to zero no further resources could be obtained until a further credit was made for more resources.

Given these approaches it would be possible for subjects to charge for resource usage according to wider rules than just the Process Base identity typically used for accounting. The account capability would be delivered as an extra parameter when calling a resource allocator in a machine capability register, rather like using a credit card in a shop.

Additional Capability Instructions

In order to enhance the ability of all software levels in the use and control of capability based access an extension to the set of capability instructions can be considered. This extension set would

parallel logical data manipulation instructions and include a set of capability indicators as shown in Fig. 5 in the primary PIR or secondary SIR indicator registers of the CPU which would be set or reset during capability instruction operations.

By providing a conditional jump (on one of up to eight capability indicators shown in Fig. 5) many, if not all, of the reluctance encountered at the simple testing level could be removed saving real time, encouraging better defensive techniques and removing special operating system facilities.

A further enhancement would include capability compare instruction(s) that enable the more dynamic tests to be performed such as the efficient searching of store for a given pointer (garbage collection) or the construction of an address and test for violation prior to use (instruction simulation).

Capability Indicators, Capability Test and Conditional Jump

The capability indicators would be set according to the first capability operand of any capability or store mode instruction. Since any instruction only sets up to seven indicators a single jump on capability indication instruction would discriminate this set of greater than eight indications (e.g. first six and last six could be instruction dependent).

The Capability Test instruction has two modes of operation. The first shown in Fig. 9 compares capability pointer values and tests for trap. The second shown in Fig. 10 tests instruction address constructions and access rights where the instruction field D is set to the required access setting (Read-Write-Execute). The block type (capability or Data) is set by the instruction into the indicators.

Masked Capability Instructions

The load and Store Capability (and compare) instruction should be maskable over the six access right bits for reducing access rights in passing or loading capability values. Masking is limited to Inform pointers only over bits 21 to 16 other mask attempts fail. The rules for masking would be conditional on the state of the most significant bit of the mask register D0. Such that it is possible to reduce Read Write Capability access to Enter access as shown in Table 2.

Consideration will now be given to the modifications necessary to the "LOAD CAPABILITY REGISTER" instruction and the "CALL" instruction to accommodate the features of the invention.

The Load Capability Instruction

When the Capability Pointer is "inform" then the Load Capability sequence is modified firstly at step S7 of the sequence shown in Fig. 6 of Patent Specification Serial No. 1,329,721, to check the Pointer Sequence (SN) Number field for zero. If SN=0 then a conventional load capability sequence is executed using C12 to access the SCT. If SN $\neq$0 then an additional capability register such as C14 which hold a Secondary Capability Table, is used. The modification steps are shown in Fig. 11. Each table can be up to 8192 words. These different capability tables can be placed into different memory modules to ease store contention. The sequence number (sn) from the limit word of the capability table must equal the Point sequence number (SN) for a successful loading to take place.

On completing the Load Capability sequence at step S21 of Fig. 6 of Patent Specification Serial No. 1,329,721 which compares the local sum check with the sum check value from the SCT entry (depicted in Fig. 11 as SC=0), when SM$\neq$0 the sequence number (SN) from the pointer is matched in step SX of Fig. 11 against the sequence number (sn) from the limit word of the table entry. Only if these three bit fields are equal is the base/limit value validly placed into the selected capability register. If the codes do not match the capability register CRX to be loaded is zeroised step SY and is therefore trapped, before entering a fault routine.

When the capability pointer loaded into the operand register OPREG (Fig. 1) is not inform (i.e. code detector CD indicates that CTS is not 11) then the load capability register micro-sequence of Fig. 6 of Patent Specification Serial No. 1,329,721 is modified at step S7 so that each of the alternative types of capability pointer may be discretely handled. Fig. 12 shows in outline the steps involved in the micro-sequence modification necessary. If the capability pointer is "inform" bits CTS will be 11 and the descriptor of the resource to be loaded into the capability register indicates an "instore" segment hence steps S8 to S21 of the micro-sequence shown in Fig. 6 of Patent Specification Serial No. 1,329,721 are performed to load the selected capability register with that descriptor. If the result of step SA however is that CTS is not 11 then CTS is tested in step SB to see if it is "Outform". If it is, the micro-sequence can make an immediate call into the trap handler to handle to transformation of the resource.

Step SC similarly handles the "data" capability pointer arrangement causing an indicator to be set whereas step SD is used to detect the passive capability pointer condition (i.e. CTS=01). When a passive capability pointer has been detected it indicates that access to internal passive

instructions is required. As mentioned previously these passive instructions are not accessible to the applications programs but are used for the handling of information within the operating system. The detection of a passive capability pointer is used in step SE to (i) set the base section of the instruction word selected capability register to zero (ii) to write the passive capability pointer value into the limit section of the selected capability register and (iii) to set up a "Trap on use" indicator.

The first operation is performed under microprogram control by activating leads CR SEL with the identity of the selected capability register while keeping the base stack BASE STK input gate G20 closed. The second operation is performed under microprogram control by opening gates G4 and G21 to circulate the passive capability pointer in the OPREG around through the arithmetic unit MILL, over the machine highway MHW to the limit stack LMT STK with leads CRSEL activated to select the required capability register. The third operation involves the priming of the trap indicator by the microprogram control unit μPROG in the secondary indicator register SIR. Exit from Fig. 12 is by way of step SF to set up the capability indicators shown in Fig. 5.

When the process specifies an instruction involving the capability register which has its base section set to zero and its limit section loaded with the passive capability pointer the trap detector TD will be activated causing the trap on use indicator to be set. This causes the trap handler process to be entered using the pointer in the limit section of the capability register to reference the trap handling table specified by hidden capability register C15 as shown in Fig. 4. Typically the trap detection arrangements may be similar to those disclosed in Patent Specification Serial No. 1,410,631.

The use of passive capability pointers accessing internal or passive instructions allows for the more secure handling of certain instructions such as the call instruction.

The Call Instruction

The Call Instruction specifies two capabilities (one a protected Enter Capability the other a functional Execute Data Capability) and performs a domain switch using the Dump Stack. The principal object is to permit the "protected function" to operate upon the machine registers D0 to D7 and C0 to C5 in some understood, approved and tested way.

In many cases a resource is created because the function is a commonly used subroutine. In other cases the effect of the operation is preponderous or devastating even though it may not be shared and it is then the protection that is a primary motivation e.g. even when the operation to be performed is of a trivial nature such as internal mode or adjust access.

Additionally the protection is both way so that the caller is protected from the subroutine. But the programmer should perhaps also be protected from himself. In other words there is also a need for explicit passing of registers over the Call/Return boundaries, rather than the passive approach used at present, requiring a series of load null capability instruction to clear any private segments.

Thirdly, the access rights to any subroutine are variable just as for any directly accessed segment. At present S-250 achieves this by creating two different enter capabilities (to different enter blocks) one has one set of executable access code blocks the other has any alternative set required.

Consequently the same resource has multiple roots each depending upon access variations. In addition, in order to protect the resource from invalid calls, its executable blocks are not offered as direct access blocks. Instead the resource protects itself by indirecting all but one executable code block. The remaining one is by convention offset '0' and merely performs parameter tests to check both normal access rights and invalid call parameters.

These three aspects of the call mechanism can be accommodated into an enhanced mode of operation that has the following new facilities. Firstly the treatment of protected instructions. Secondly the explicit indication of registers to be passed into the subroutine and thirdly access right checking by the call instruction.

Protected Instructions

When the protected function aspect of an enter capability is required it can be achieved for simple operations by replacing the normal Inform Enter Capability with an Executable Passive Capability. For example "Set Interrupt Inhibit" or "Load Capability Registers Range C10 to 17" (protected at present globally by Internal Mode) could be individual instructions implemented by "Passive Enter" Capabilities of a special "Executable Access Type", operated by the call instruction. Thus the protected function is called into operation and has all the protection afforded normal Enter Resources.

Explicit Register Passing

If calls via C_0—C_5 and limited to a 3 bit offset. The remaining 6 bits (the Pass Field) of the 9 bit offset could be a linear coding which relates to C_0—C_5 and D_0—C_5. Only

when a pass bit is set is the corresponding register passed into the subroutine. When a bit is not set the corresponding capability register and data register is reloaded with a null value during the call. The return Instruction operates in a similar way using the same bit coding of the Literal Field.

The simplest and sufficient approach is for the hardware to discard any values service in that way. This demands that software saves any dynamically acquired blocks not preserved over the Call/Return interface. All calls via C_6 would not need to use the facility and could therefore retain a 9 bit offset for the call instruction.

Subroutine Access Variations

If the integration of access code significance is steered by the Enter type and the call instruction then the bits other than enter can hold a dual significance. In one state they would retain their current significance. In the other state they would enable a particular call offset.

Consider therefore the Enter Capability Access Right (bit 21) as a more generalised "Low Level/High Level Access Status Indicator". In the Low Level State (set bit 21 to zero) the Capability Access bits (bits 16—20) retain their current significance, namely Read Data, Write Data, Execute, Read Capability and Write Capability.

When the High Level state is set (bit 21 to one) then bits 16 through 20 do not possess any low level significance. Thus if bit 21 is set and bit 16 is set then Read Data is *not* permitted. As at present when bit 21 is set the call instruction specifies which offset to call and, within the limitations of offsets 0 to 4, the bits 16 to 20 *if set* will permit access to take place i.e. the call with occur.

On executing the call the high level access code in total must be reset (bits 16 to 20 *if set* will permit access to take place i.e. the call will occur).

On executing the call the high level access code in total must be reset (bits 16 to 21 all set to zero), and then low level Read Capability access should be set (set bit 19 as at present). For more complex situations the previous example can still be implemented with fan out code etc. Moreover since it will no longer be possible to have Enter Access and Read Access it should be made possible to call any Read Capability blocks, since this will permit recursive calls and does not present any new threats to performance or security. Calls via C_6 can be made to any offset within C_6.

The Interrupt Mechanism

As a result of the modifications necessary to achieve the invention there is a corresponding need to modify the interrupt mechanism. The alternative actions are shown in Fig. 13 for the various types of interrupt action. The type φ call action would perform a Trap call to offset zero of the trap table. This entry would be reserved for the trap handler and not available for a passive type code. The passive type code zero would instead be used by directly executable passive type for example an implied capability as shown in Fig. 14.

WHAT WE CLAIM IS:—

1. A data processing system including a central memory and at least one processor unit incorporating information protection techniques involving capability addressing arrangements of the type having a system capability table having one entry for each resource storing descriptor information indicative of the location of that resource and each process is provided with at least one reserved segment pointer table comprising a list of capability pointers, each pointer including a resource type indicator an access code and a pointer value and the resource type indicator defines (i) an inform resource, (ii) an outform resource or (iii) a passive resource and wherein when a capability pointer having a passive resource indicator is detected during a load capability register operation the capability register to be loaded is conditioned such that an automatic trap operation is performed when the said capability register is used causing entry into a trap handling process through a passive resource table, the descriptor for which is held in one of the hidden capability registers, and the pointer value of the passive capability pointer is used to define a passive resource table entry for conditioning the processor to perform a particular passive resource handling process.

2. A data processing system according to claim 1 in which one of the passive resource handling processes involves allocation and adjustment of capability access codes.

3. A data processing system according to claim 1 or 2 in which one of the passive resource handling processes involves the movement of object segments into and out of the central memory.

4. A data processing system according to claim 1, 2 or 3 in which one of the passive resource handling processes involves the transformation or deletion of capability pointers.

5. A data processing system according to any one of the preceding claims in which one of the passive resource handling processes involves the creation of system capability table entries.

6. A data processing system as claimed in claim 1 in which the or each processor unit includes (a) means for loading at least part of a capability register with a discrete

* C L M S *

when a pass bit is set is the corresponding register passed into the subroutine. When a bit is not set the corresponding capability register and data register is reloaded with a null value during the call. The return Instruction operates in a similar way using the same bit coding of the Literal Field.

The simplest and sufficient approach is for the hardware to discard any values service in that way. This demands that software saves any dynamically acquired blocks not preserved over the Call/Return interface. All calls via C_6 would not need to use the facility and could therefore retain a 9 bit offset for the call instruction.

Subroutine Access Variations

If the integration of access code significance is steered by the Enter type and the call instruction then the bits other than enter can hold a dual significance. In one state they would retain their current significance. In the other state they would enable a particular call offset.

Consider therefore the Enter Capability Access Right (bit 21) as a more generalised "Low Level/High Level Access Status Indicator". In the Low Level State (set bit 21 to zero) the Capability Access bits (bits 16—20) retain their current significance, namely Read Data, Write Data, Execute, Read Capability and Write Capability.

When the High Level state is set (bit 21 to one) then bits 16 through 20 do not possess any low level significance. Thus if bit 21 is set and bit 16 is set then Read Data is *not* permitted. As at present when bit 21 is set the call instruction specifies which offset to call and, within the limitations of offsets 0 to 4, the bits 16 to 20 *if set* will permit access to take place i.e. the call with occur.

On executing the call the high level access code in total must be reset (bits 16 to 20 *if set* will permit access to take place i.e. the call will occur).

On executing the call the high level access code in total must be reset (bits 16 to 21 all set to zero), and then low level Read Capability access should be set (set bit 19 as at present). For more complex situations the previous example can still be implemented with fan out code etc. Moreover since it will no longer be possible to have Enter Access and Read Access it should be made possible to call any Read Capability blocks, since this will permit recursive calls and does not present any new threats to performance or security. Calls via C_6 can be made to any offset within C_6.

The Interrupt Mechanism

As a result of the modifications necessary to achieve the invention there is a corresponding need to modify the interrupt mechanism. The alternative actions are shown in Fig. 13 for the various types of interrupt action. The type φ call action would perform a Trap call to offset zero of the trap table. This entry would be reserved for the trap handler and not available for a passive type code. The passive type code zero would instead be used by directly executable passive type for example an implied capability as shown in Fig. 14.

WHAT WE CLAIM IS:—

1. A data processing system including a central memory and at least one processor unit incorporating information protection techniques involving capability addressing arrangements of the type having a system capability table having one entry for each resource storing descriptor information indicative of the location of that resource and each process is provided with at least one reserved segment pointer table comprising a list of capability pointers, each pointer including a resource type indicator an access code and a pointer value and the resource type indicator defines (i) an inform resource, (ii) an outform resource or (iii) a passive resource and wherein when a capability pointer having a passive resource indicator is detected during a load capability register operation the capability register to be loaded is conditioned such that an automatic trap operation is performed when the said capability register is used causing entry into a trap handling process through a passive resource table, the descriptor for which is held in one of the hidden capability registers, and the pointer value of the passive capability pointer is used to define a passive resource table entry for conditioning the processor to perform a particular passive resource handling process.

2. A data processing system according to claim 1 in which one of the passive resource handling processes involves allocation and adjustment of capability access codes.

3. A data processing system according to claim 1 or 2 in which one of the passive resource handling processes involves the movement of object segments into and out of the central memory.

4. A data processing system according to claim 1, 2 or 3 in which one of the passive resource handling processes involves the transformation or deletion of capability pointers.

5. A data processing system according to any one of the preceding claims in which one of the passive resource handling processes involves the creation of system capability table entries.

6. A data processing system as claimed in claim 1 in which the or each processor unit includes (a) means for loading at least part of a capability register with a discrete

characteristic code in response to the detection of a passive resource type indicator and (b) discrete characteristic code detection means arranged to monitor the information content of a capability register as it is used.

7. A data processing system as claimed in claim 1 in which information is stored in segments in the storage units and each information segment comprises a segment header and a user object segment and inform pointer controlled references to the information segment involve access to the user object segment only whereas when an operating system process is being executed means are operable in the processing unit to confine manipulative access to the segment header of a segment.

8. A data processing system according to claim 7 in which the segment header includes information indicative of the type of user object segment and its location on a backing store.

9. A data processing system as claimed in claim 7 or 8 in which when an operating system process is being executed means are operable in the processor unit to permit read-only access on a serial transfer basis to the complete information segment.

10. A data processing system as claimed in claim 7, 8 or 9 in which access to the information segment by an operating system process is performed using a passive capability pointer.

11. A data processing system as claimed in claim 1 in which each inform capability pointer includes a system capability table sequence number and a pointer value and a plurality of system capability table sections are provided each section being identified by a discrete sequence number and the pointer value is arranged to define an entry in the table section identified by the accompanying sequence number.

12. A data processing system according to claim 11 in which the central memory comprises a plurality of storage modules and each system capability table section is arranged to be held in a different storage module.

13. A data processing system according to claim 11 or 12 in which each system capability table entry includes a sequence number and the load capability register operations include steps for comparing the sequence number held in the inform capability pointer with that held in the system capability table entry accessed.

14. A data processing system according to claim 1 in which system subroutine call operations are performed using inform capability pointers which include an enter access right code characterised in that subroutines which are common to a plurality of processes are provided with a common enter capability segment defined by the pointer value in the inform capability pointer and each inform enter capability pointer includes a plurality of enter access bits and each enter access bit when in a first state is indicative of the right to use a given linear offset of the common enter capability segment whereas when an enter access bit is in a second state it is indicative that the calling process does not have the right to use the so defined offset and the processor unit includes means for checking each subroutine access to an enter capability segment entry.

15. A data processing system substantially as described and as shown in the accompanying drawings.

R. J. HART,
Chartered Patent Agent,
For the Applicant.

Printed for Her Majesty's Stationery Office, by the Courier Press, Leamington Spa, 1981
Published by The Patent Office, 25 Southampton Buildings, London, WC2A 1AY, from which copies may be obtained.