Appendix A
Object-Oriented Addressing

The previous chapters introduced Data General's new GALAXY architecture and its GALAXY 1000 implementation. This appendix describes the 1000 system's addressing mechanisms in more detail. We also demonstrate how the system's protection mechanism functions in parallel with the addressing mechanisms. We include a simple example of how the system develops a FORTRAN variable's main memory address.

Basic Addressing and Protection Operations

Figure A-1 below shows the 1000 computer's unaccelerated addressing flowchart. This flowchart shows the system's three major addressing functions:

- Namespace Addressing
- Logical Address Reduction
- Logical Address Translation

The chart also shows the system's protection checking operation. These addressing and protection checking operations are performed by the Job Processor's Fetch Unit (F-unit).

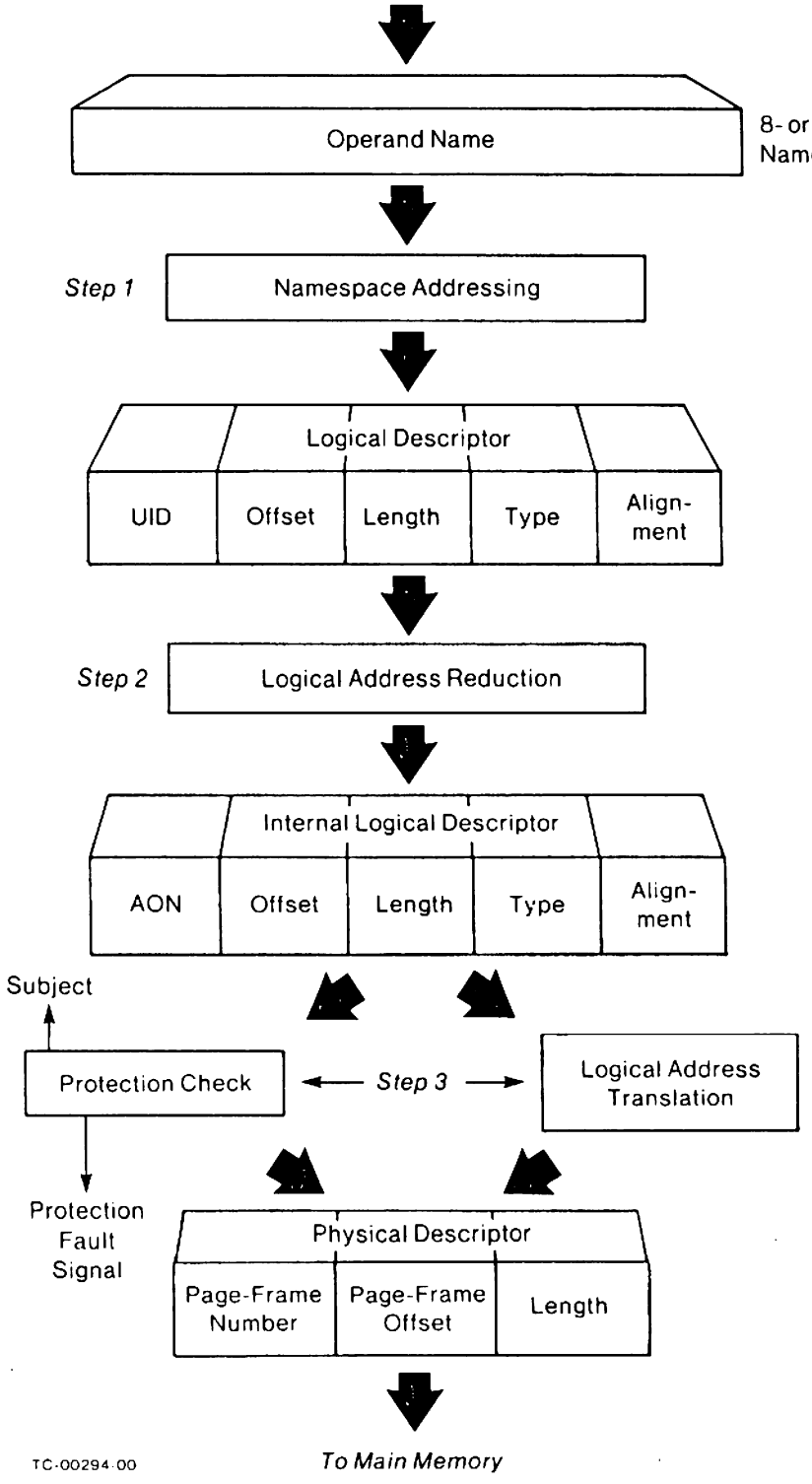


Figure A-1. The 1000's Basic Addressing Flow

Namespace Addressing (Step 1 in Figure A-1) is the key to the system's addressing flexibility and performance. Namespace Addressing performs calculations that transform S-language operand names into logical descriptors. The system uses logical descriptors to locate operands and other data structures within objects. Our description of the addressing process assumes that we are referencing an operand from an S-language instruction stream. The part of a logical descriptor which locates an operand is its logical address (UID and offset). A logical descriptor also specifies an operand's length, and, optionally, its type and alignment.

In Step 2, Logical Address Reduction (LAR) associates an object's Unique Identifier (UID) with an Active Object Number (AON). This operation creates an internal logical descriptor; a descriptor that identifies an object with an AON instead of an UID. An object is considered to be *active* if the system can reference it with an AON. AONs are 2^{14} bits long, which means that the 1000 can address up to 16,384 active objects directly. LAR uses a simple table look-up if the referenced object is active, or kernel operating system routines if the object is not active.

Logical Address Translation (Step 3) converts a logical descriptor into its corresponding physical descriptor. A physical descriptor contains a main memory address, and length and alignment information. A main memory address consists of a page-frame number and a starting offset in the page frame.

The F-unit performs protection checking operations (part of Step 3) in parallel with Logical Address Translation. Protection faults result if a user does not have proper access rights to a particular object. The system automatically suspends any memory reference that produces a protection fault.

The next four sections describe the 1000 system's addressing and protection checking mechanisms. We conclude the description with a simple example of how these operations work together. Remember that the F-unit has three caches that speed up (accelerate) these basic operations. We describe accelerated addressing and protection checking at the end of this appendix.

Namespace Addressing

We said that conventional computers generally offer a limited number of basic addressing modes. The number of addressing modes is limited by the number of addressing mode bits available in a system's machine language instruction. Conventional addressing modes usually include absolute, Program Counter relative, and accumulator relative addressing. They also include direct, indirect, and indexed addressing.

Namespace Addressing provides all of these conventional modes and more. In fact, Namespace Addressing lets GALAXY computers address many kinds of complex data structures, such as arrays and records of arrays. Namespace Addressing provides this flexibility because S-language instructions do not contain operand addressing information. S-language instructions carry only operation codes and operand names. An operand's addressing information is separated during compilation and placed in a Name Table.

Figure A-2 below illustrates a simple example of a Namespace Addressing calculation. The illustration shows how an S-language operand name indexes a program's Name Table to produce a Name Table Entry (NTE). An NTE contains all of the information that Namespace Addressing needs to calculate a data structure's logical address.

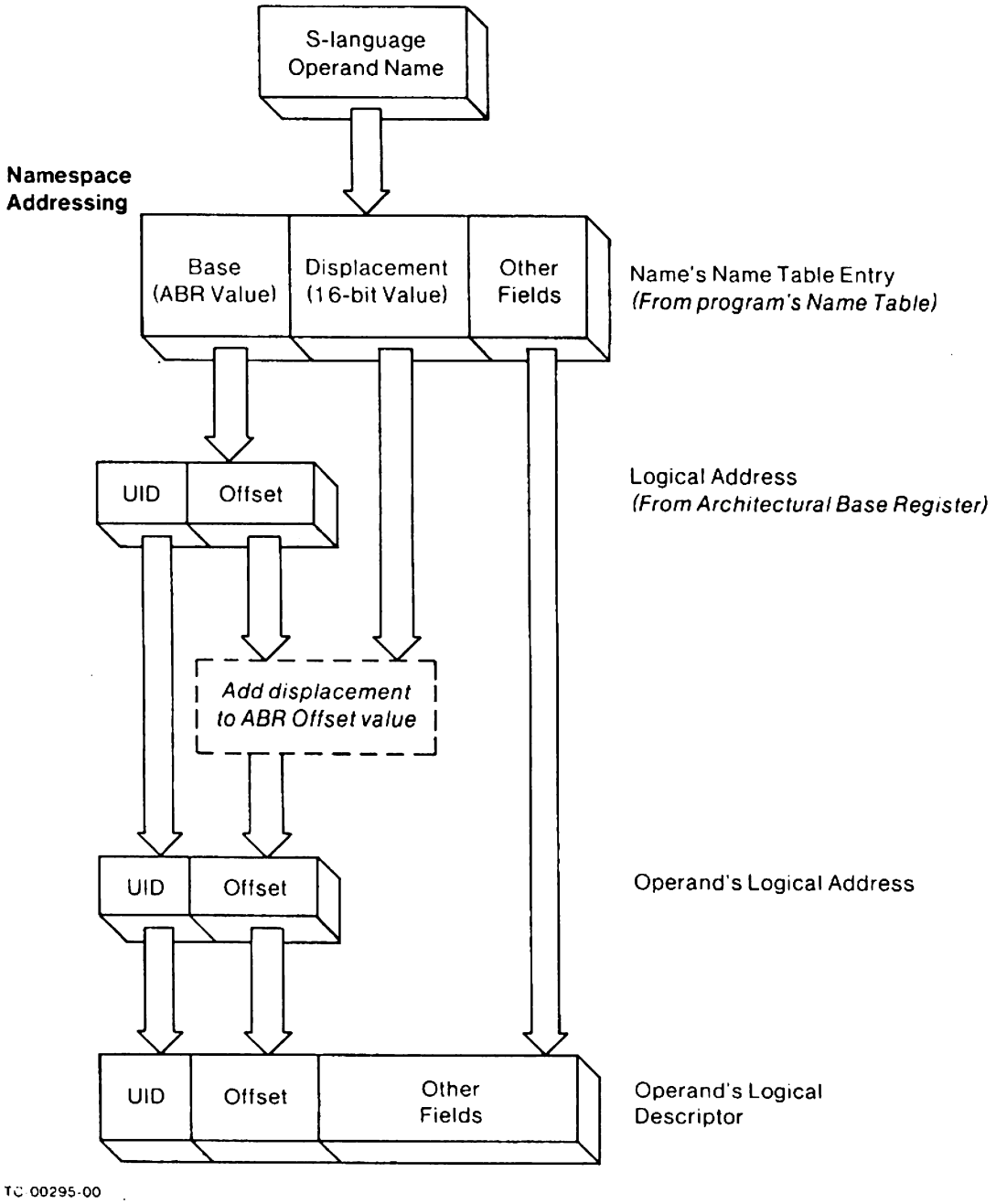


Figure A-2. Simplified Namespace Addressing

Namespace Addressing uses base and displacement calculations to determine a data structure's logical address. A calculation's base value is a logical address (UID and offset), which is taken from one of three Architectural Base Registers (ABRs). The operating system manages these ABRs —they are invisible to high-level language programmers. Each ABR contains a logical address that locates one of a process' data storage structures (such as a stack, which contains information used by the process). ABR values are set when a process is created. The values are changed automatically by certain operations, such as procedure calls and returns.

Displacement is a signed integer value, which is added to the base's offset to produce a new logical address. Namespace Addressing also provides recursive addressing—that is, some NTE fields can refer to other NTEs. For example, an NTE's base field can contain a name instead of an ABR's identification number.

Namespace Addressing has two basic operations: name resolution and name evaluation. *Resolving* a name returns its logical descriptor (address, length, and type). *Evaluating* a name returns its current value, length, and type.

Name Table Entries have two formats: short (64 bits) and long (128 bits). Short NTEs provide information needed to address most simple data structures. The system uses long NTEs to address more complex structures, such as arrays. Long NTEs have an extended displacement field, which the system uses to reference structures with addresses greater than 2^{16} bits into an object. Namespace Addressing can locate elements of multi-dimensional arrays by using two long NTEs for each dimension. For example, the system can address elements of a three-dimensional array with the information contained in six long NTEs. One Name Table can hold up to 2^{16} short NTEs, or 2^{15} long NTEs, or a mixture of both.

Changing a logical descriptor's UID to an AON creates an internal logical descriptor. An internal logical descriptor contains the same information as a full descriptor, except that the smaller, reusable AON is substituted for the descriptor's UID.

The F-unit's data paths can transfer and manipulate internal logical descriptors directly (see Figure A-3). Note that LAR does not compress the offset component of the logical address. The 1000 system can reference any bit in an active object.

LAR uses two operating system tables as data bases: the Active Object Table and the Active Object Hash Table. An object is active when it has an entry in the Active Object Table (AOT). The system gets information for AOT entries from the Logical Allocation Unit Directory (LAUD), which contains the object. Each AOT entry matches an AON with an UID. AOT entries also contain frequently used attribute information (such as an object's length and

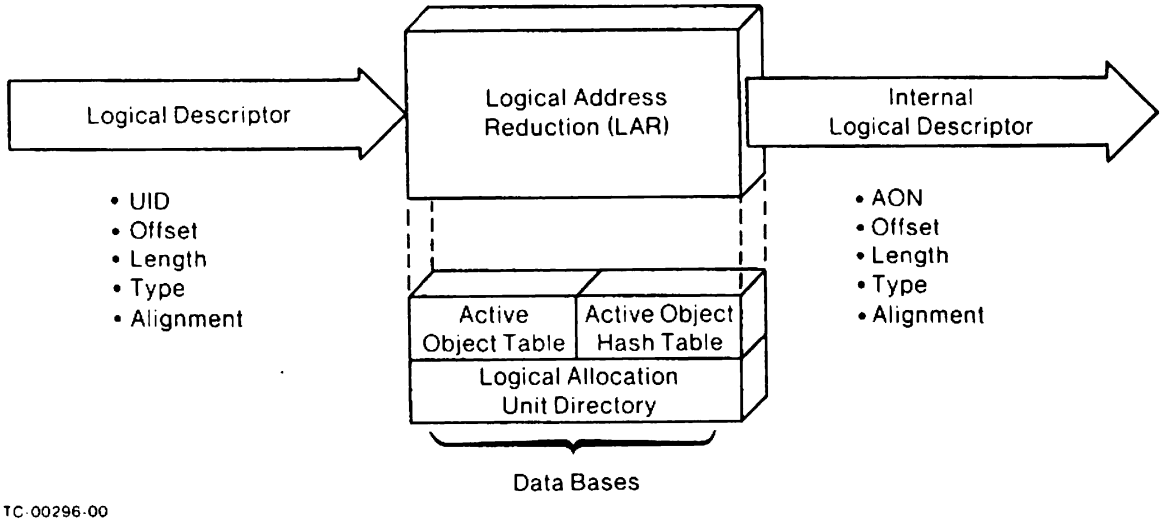


Figure A-3. Logical Address Reduction

Logical Address Reduction

Logical Address Reduction (LAR) associates an object's 80-bit UID with a smaller internal identification number. This more manageable identifier is called an Active Object Number (AON). AONs are unique to a system; the system reassigns AONs as it needs them. Some of a system's AONs are permanently assigned to operating system data base objects and data structures.

type). The operating system lets you set the size of both tables during system generation, to suit your system's specific requirements.

Searches through the AOT are accelerated by hash-coding an object's 80-bit UID and using the hashed value as an index into the Active Object Hash Table (AOHT). An AOHT entry contains an Active Object Number, which the system uses to index the

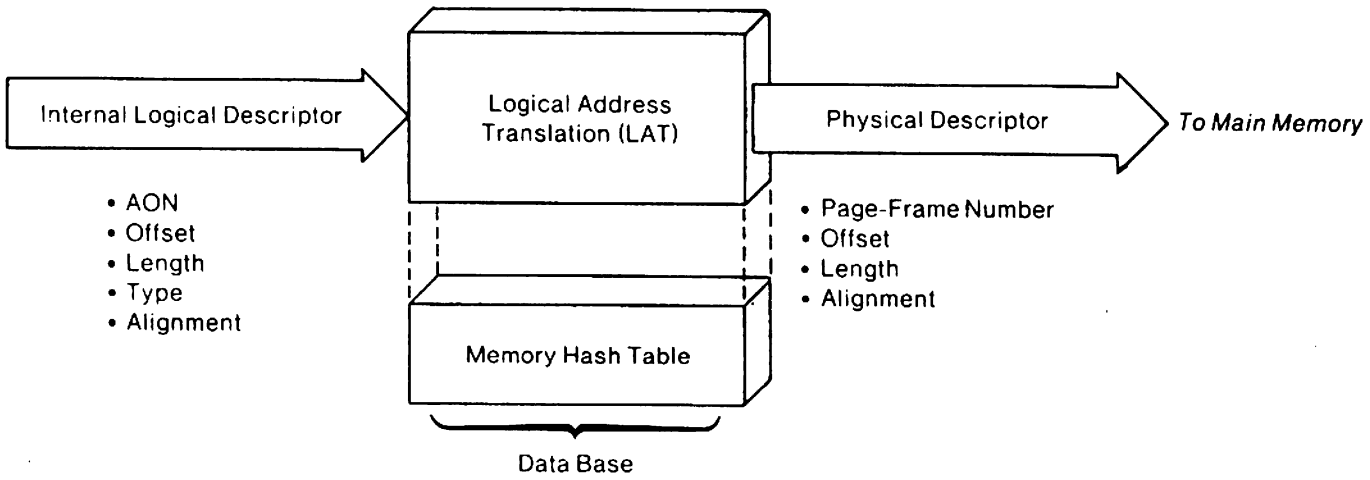


Figure A-4. Logical Address Translation

AOT. AOT entries contain pointers (called *threads*) to other AOT entries that have the same hash code. The operating system automatically traverses a group of threaded entries until it finds an entry with the requested UID, or reaches the end of a thread. If the system reaches the end of a thread without finding an entry with the proper UID, the object is not active.

The system uses a series of microprogrammed routines to activate objects. These routines deactivate the least recently referenced object by removing its Active Object and Active Object Hash Table entries. The deactivated object's attributes are put back into their proper Logical Allocation Unit Directory. The system activates a newly referenced object by placing its UID and selected attribute information into the Active Object Table. This new AOT entry contains the referenced object's UID-to-AON relationship, and the system can reference the object by using its Active Object Number.

Logical Address Translation

The last addressing step is Logical Address Translation (LAT). Recall that the 1000 operating system divides objects into pages, to manage its main memory resource efficiently. What we didn't describe was how systems convert logical addresses into physical addresses. This conversion is performed by the LAT operation, shown in Figure A-4 below.

LAT accepts an internal logical descriptor and produces the referenced operand's physical descriptor. A physical descriptor includes an operand's physical address and length. Because 1000 objects are paged, physical addresses are expressed as page-frame numbers and offsets in a page frame. The system automatically handles situations in which the referenced data structure crosses page-frame boundaries. A memory reference terminates if the system's protection check shows that the requestor does not have the proper access rights to the referenced object.

LAT uses the Memory Hash Table (MHT) as its data base. An MHT entry contains information specifying which logical page is currently associated with a page frame (see Figure A-5 below). Each entry has an AON and page number field that represents an internal logical address, and a page-frame number field that represents the corresponding physical address.

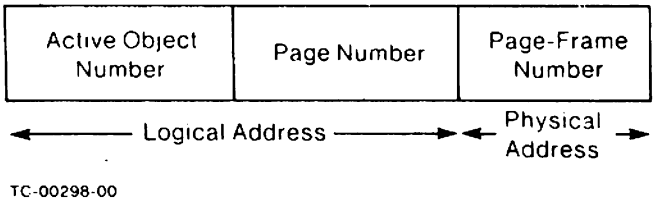


Figure A-5. Memory Hash Table Entry

Protection Checking

Determining a subject's access privileges quickly is important to system performance, because the system checks access rights each time an object is referenced. The 1000 system performs protection checking in parallel with Logical Address Translation. The system's basic protection checking operation is shown in Figure A-6 below.

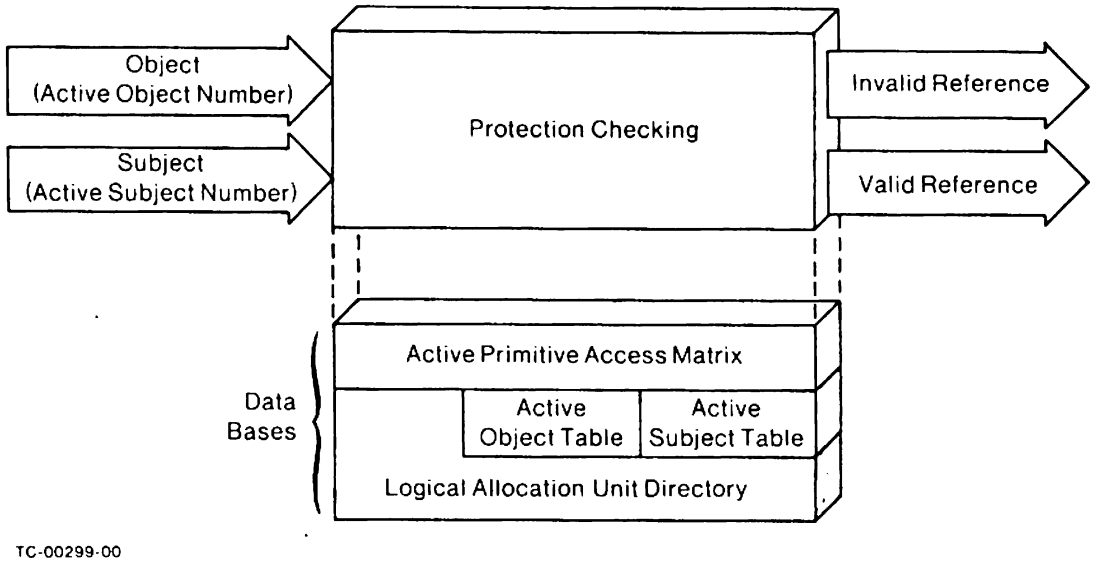


Figure A-6. Protection Checking

GALAXY systems use Unique Identifiers (UIDs) to identify system users (system users are called subjects). The 1000 operating system temporarily assigns a smaller 12-bit Active Subject Number (ASN) to frequently used subjects. Like Active Object Numbers, ASNs are smaller and easier to manage than 80-bit UIDs. Information about each active subject is contained in Active Subject Table (AST) entries. An AST entry matches an active subject's UID with its current Active Subject Number.

An active subject's rights to access an active object are contained in a system table called the Active Primitive Access Matrix (APAM). The APAM (shown in Figure A-7) has the system's active subjects as one axis and its active objects as the other.

The APAM is indexed by active subject and active object to obtain a subject's access rights to a referenced object. An object's access information is taken from its Access Control List (ACL). The system uses the APAM to avoid searching through a

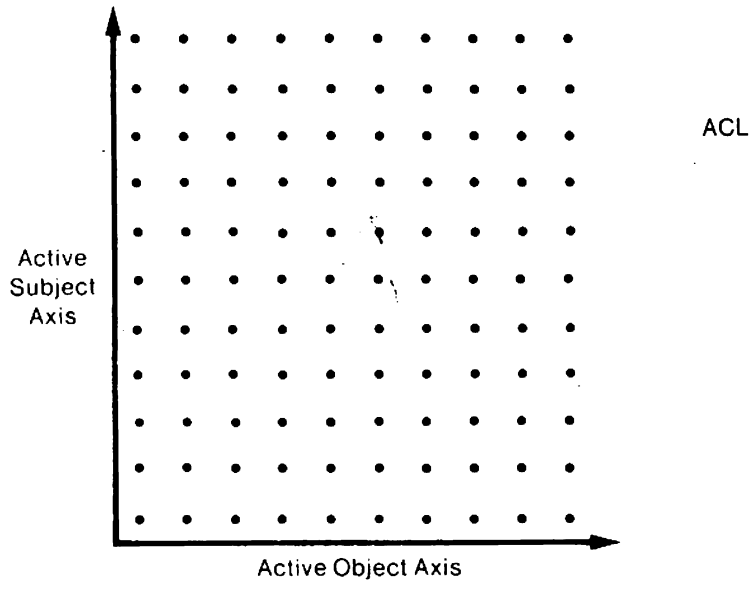


Figure A-7. Active Primitive Access Matrix

Logical Allocation Unit Directory to find an object's ACL.

A protection check can fail if a subject does not have proper access rights to an object. The check can also fail if a subject attempts an operation that reads or writes past an object's boundaries. When a check fails, the operating system terminates the memory reference and sends the subject an error message.

Name Table Entry Interpretation

The first addressing step is to remove the operand's numerical name from the S-language instruction. The system's FORTRAN compiler creates the name and corresponding Name Table Entry (NTE). The compiler also places the operand's NTE into the program's Name Table. We show the S-language expansion for this example FORTRAN statement in Figure A-8 below. In this example, the compiler has assigned the numerical name 321 to the variable 'N'.

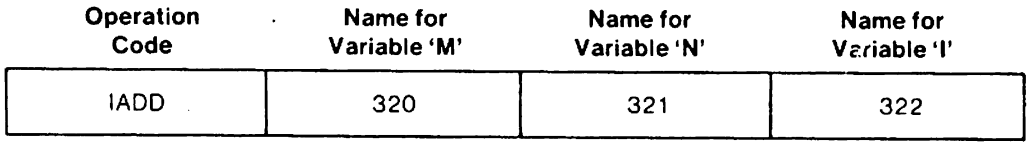


Figure A-8. Names in the S-instruction

An 1000 Addressing Example

This section illustrates the 1000's addressing and protection checking operations introduced above. Our example starts with a simple FORTRAN statement and shows how the system locates one of the statement's variables.

The FORTRAN statement `I = M + N` adds the integer variables 'M' and 'N' and puts the result into 'I'. The FORTRAN S-language expansion for this statement is:

```
IADD M N I
```

where 'M', 'N', and 'I' are represented by numerical names assigned by the compiler. The IADD operation code specifies a three-operand integer add.

Our example describes how the system finds the N variable in main memory. You may want to refer back to Figure A-1 as you read this example.

The system needs to know the length of the operand names in the procedure's S-language instructions, so that it can separate opcodes and names. The 1000 system's FORTRAN compiler uses 8-bit or 16-bit names and 8-bit operation codes. Therefore, the system can easily separate S-language operation codes and operand names. The S-language Program Counter (PC) locates the currently executing S-language instruction. The PC always points to the beginning of an instruction's operation code. The PC is automatically incremented by the number of bits in an instruction when the processor executes the instruction.

Namespace Addressing examines name 321's NTE and performs the calculations needed to produce the operand's logical descriptor. The NTE has an information (flags) field that the operating system uses to determine which "addressing mode" to use to locate the operand. The flags field also specifies which of the three ABRs contains the operand's base logical address. We'll assume that our example's NTE flags field indicates that 'N' is a 16-bit integer variable. The NTE is shown in Figure A-9 below.

Flags and Alignment	Type	Base	Length	Displacement
• Short NTE • Right-justified • Base is an ABR	Integer	Static Data Pointer	16 bits	18192 bits

TC-00302-00

Figure A-9. A Name Table Entry

The system uses the Name Table Entry's base and displacement information to calculate the operand's logical address (UID and offset). The base for our example is the logical address in the Static Data Area's ABR. This means that the operand's value persists across all activations of the program. The NTE's displacement value indicates that the operand is located 18,192 bits from the Static Data Area's current location. Assume that the logical address in the ABR is [1741,1024]. This means that this Static Data Area begins at bit number 1024 in object 1741. Therefore, variable N's starting location is at (1024 + 18192) = 19,216 bits into object 1741.

The logical descriptor for our example operand is shown in Figure A-10 below. Note that NTE information about the operand that is not used in the base and displacement calculations (length, type, and alignment) is passed through to the descriptor.

Logical Address Reduction

Logical Address Reduction (LAR) now determines if object 1741 is currently active. LAR sends UID 1741 through a hashing operation that compresses the UID into a small hash code value. The hash code value is used as an index into the Active Object Hash Table. In our example, the AOHT entry for the hash code of 4 contains Active Object Number 748 (see Figure A-11).

Flags and Alignment	Type	UID	Offset	Length
Right-justified	Integer	1741	19126 bits	16 bits

TC-00303-00

Figure A-10. Logical Descriptor for the Variable 'N'

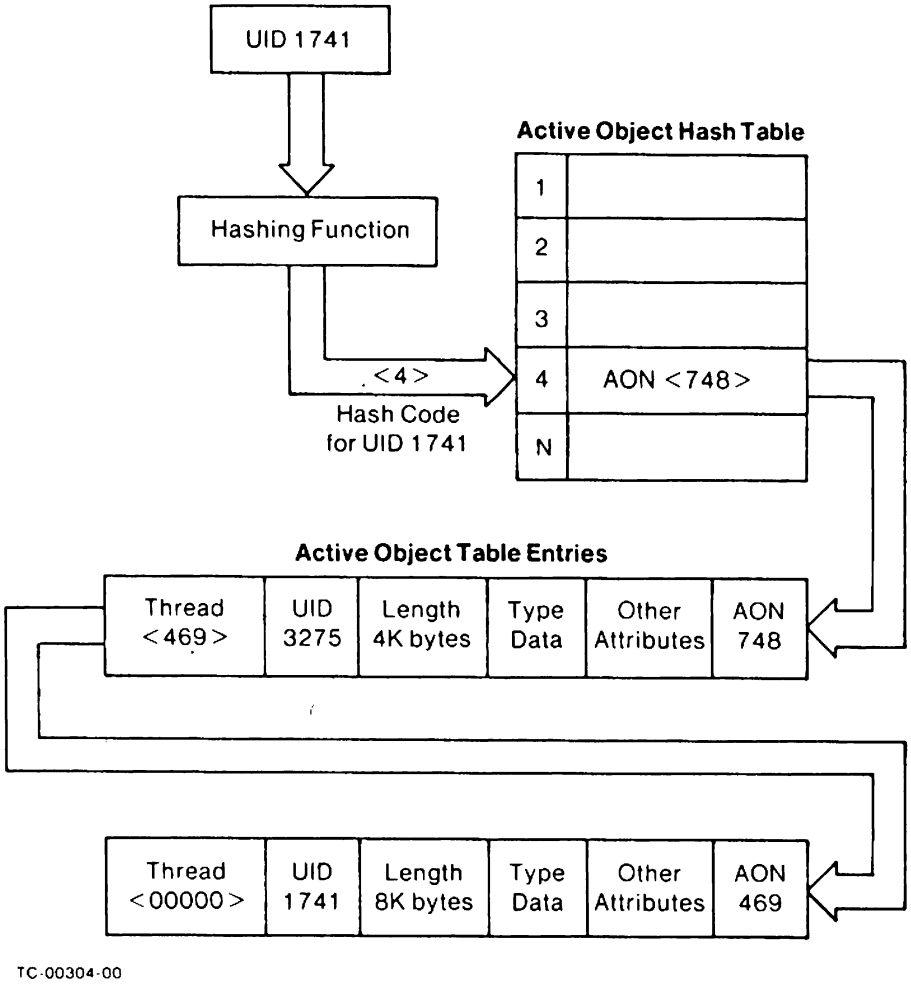


Figure A-11. A Logical Address Reduction

The system indexes the Active Object Table at the entry for AON 748 and compares the referenced UID (1741) to the UID value in the AOT entry. This comparison is necessary, because more than one UID can be hashed to the same hash code number (called a collision). AOT entries have threads to other entries with the same hash code. Our example shows that UID 3725 has the same hash code (4) as UID 1741, so the first UID comparison fails. However, the UIDs match when the system follows the thread to the next entry with the same hash code. The null thread (all zeros) in entry number 469 means that this is the last entry for this particular thread.

The UID-to-AON conversion would have failed if UID 1741 had no Active Object Table entry. This would mean that the referenced object was inactive, or did not exist. If the object is not active, the system automatically deactivates the least recently used object so that its AON may be reassigned. The deactivated object's UID and attributes are removed from the AOT and replaced with the requested object's UID and attributes. The system generates an error message if the object does not exist.

Figure A-12 below illustrates the internal logical descriptor which would be produced for our example. Notice that no reduction is performed on the operand's starting location (the offset) in the object.

Flags and Alignment	Type	AON	Offset	Length
Right-justified	Integer	469	19216	16

TC-00305-00

Figure A-12. Internal Logical Descriptor for the Variable 'N'

Logical Address Translation

The system now performs the final addressing step: Logical Address Translation (LAT). LAT converts the descriptor's logical address (AON and offset) into a physical address (shown in Figure A-13 below). The physical address becomes part of the operand's physical descriptor. This physical descriptor contains the information required to access the operand in the system's main memory. The descriptor contains a main memory page-frame number, the operand's starting offset in that page frame, and the operand's length. This offset reflects the offset in the page frame, not in the object.

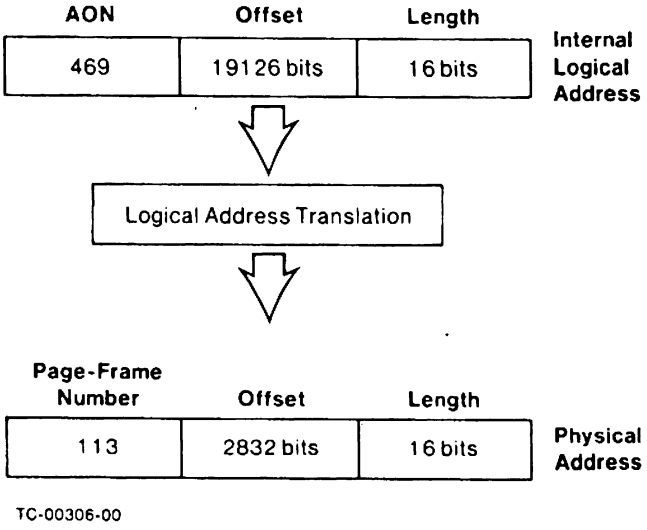


Figure A-13. A Logical Address Translation

The example operand's starting location is 19,216 bits into the object. The system calculates an operand's physical location (page number and bit-on-page offset) by dividing the operand's starting location by the number of bits in a page (16,384). The quotient determines a page number; the remainder determines the bit-on-page offset. Therefore, our example operand starts at bit number 2832 of the object's first page ((19,216/16,384) = 2832 bits). Figure A-14 shows the location of the 'N' variable in a 1000 system's main memory.

Protection Checking

The system indexes the Active Primitive Access Matrix (APAM) to see if the matrix contains an entry for the proper subject and object. The proper APAM entry will have the current subject's access rights to the object. Assume that the process that is executing our FORTRAN program is being run for

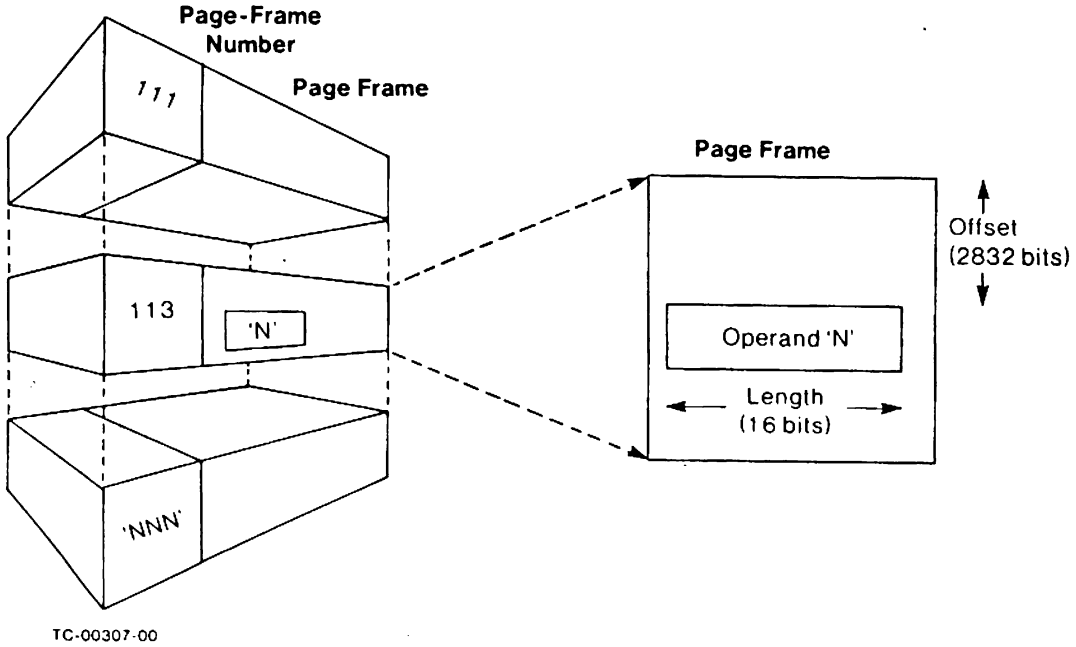
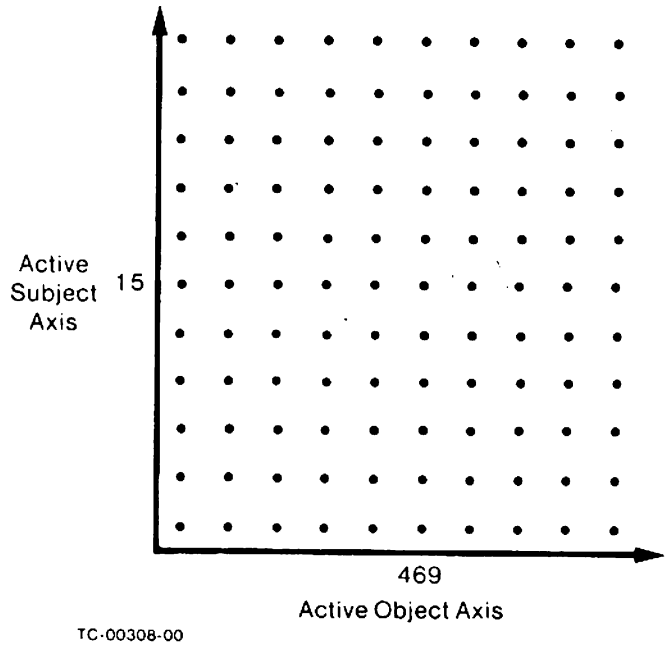


Figure A-14. The 'N' Operand in Main Memory

the the subject Jones, and that Jones' Active Subject Number (ASN) is 15. When the system indexes the APAM by AON 469 and ASN 15, it finds that Jones has read and write privileges to the object (see Figure A-15 below).



Read	Write	Execute	Non-Data Modes
1	1	0	0

Figure A-15. APAM Example

The system uses access information from the Logical Allocation Unit Directory (LAUD) if the APAM has no entry for object 469. Because Jones can read object 469, the system can move the current value of 'N' into the Job Processor's Execute Unit (E-unit). The E-unit then receives the value of 'M', adds 'M' and 'N', and sends the result back to the F-unit.

Accelerated Addressing With the F-unit Caches

The preceding example described the F-unit's unaccelerated address and protection operations. However, the F-unit has three caches that accelerate its addressing and protection checking operations. These caches are the Name Cache, Address Translation Cache, and Protection Cache. The Name Cache accelerates Namespace Addressing calculations. The Address Translation Cache accelerates Logical Address Translation, and the Protection Cache accelerates protection checking operations.

Figure A-16 illustrates a 1000 system's accelerated addressing flow. This flow chart represents the fastest and most commonly used path through the 1000's addressing and protection operations (based on typical cache hit-rates of 85 to 90 percent). We describe the functions and data structures shown in this flowchart in the following sections.

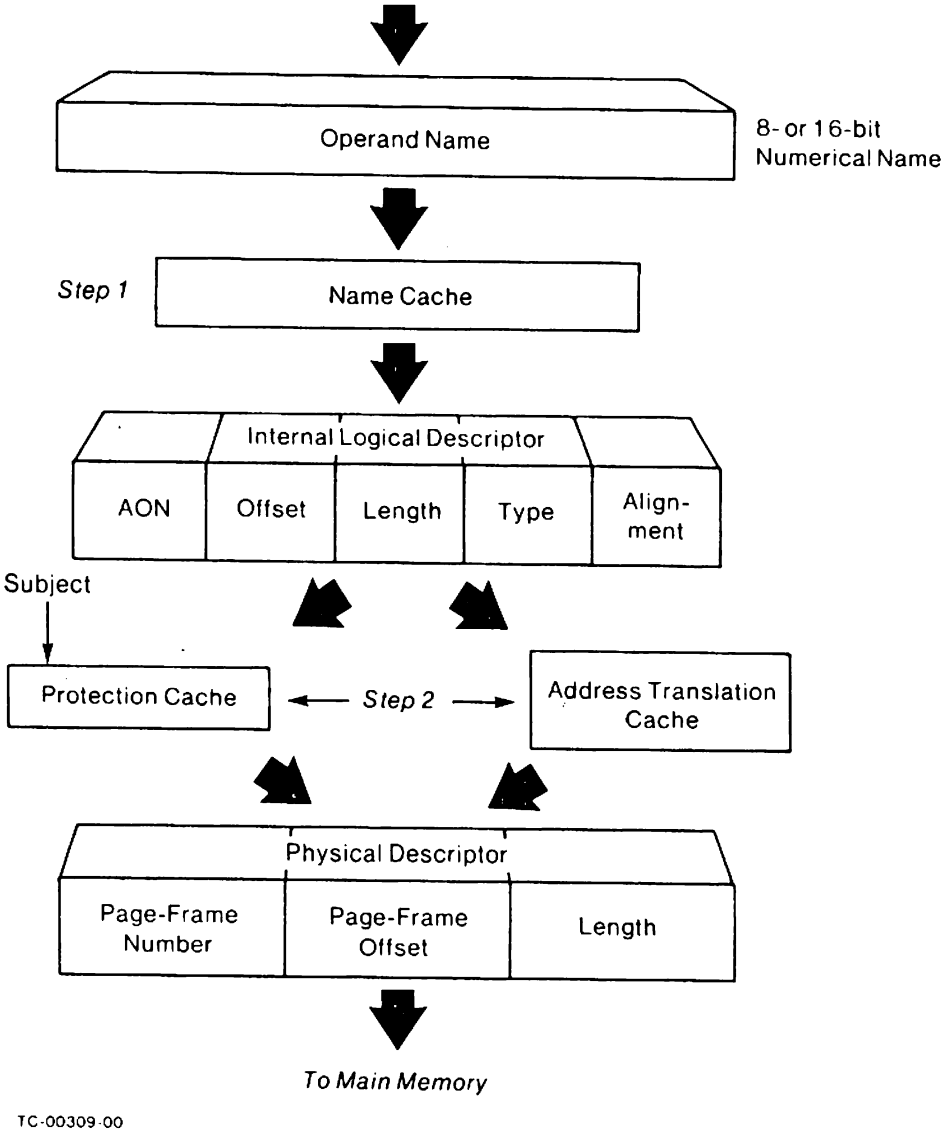


Figure A-16. Addressing With the F-unit Caches

Name Cache

We said earlier that Namespace Addressing uses base and displacement calculations to determine an operand's starting address. The information needed to calculate an operand's address is contained in the name's Name Table Entry (NTE). The 1000 system's Name Cache stores this information for many kinds of recently used names. The Name Cache can store information for up to 64 names.

The system always presents an operand name to the Name Cache first. A cache "hit" occurs if the name's internal logical descriptor is available in the cache. This descriptor includes the variable's Active Object Number (AON), offset into the object, length, type, and alignment. A Name Cache hit lets the system bypass a Name Table look-up, NTE interpretation, and Logical Address Reduction (LAR). LAR is bypassed because the Name Cache stores internal logical descriptors. These descriptors already identify objects with an AON instead of an UID.

In our example, the system uses the name of the variable 'N' (321) as an index into the Name Cache. A Name Cache entry produces the variable's internal logical descriptor directly.

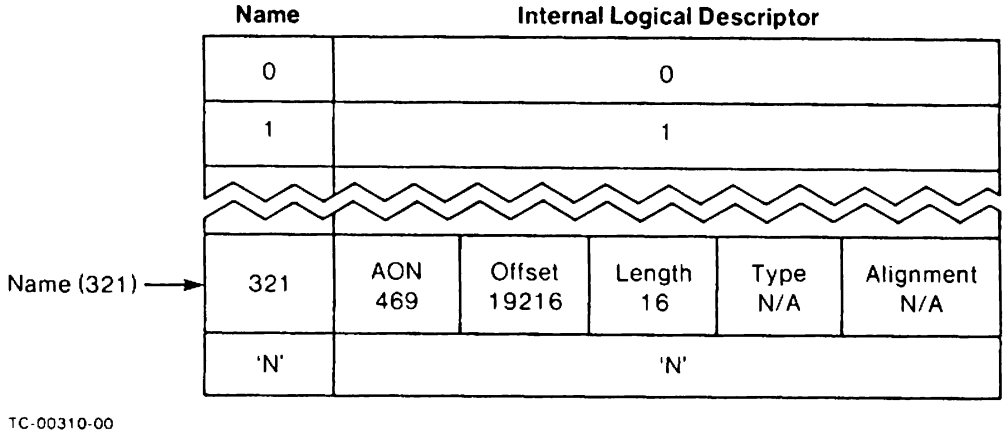


Figure A-17. The Name Cache

If the requested name is not in the Name Cache, the system removes its NTE from the Name Table and interprets it to produce the name's logical descriptor. Then, Logical Address Reduction replaces the logical descriptor's UID with an Active Object Number to produce the internal logical descriptor.

Address Translation Cache

The system uses the Address Translation Cache to bypass Logical Address Translation. The Address Translation Cache (ATC) contains information that directly associates a logical descriptor's AON, offset, and length with a physical descriptor (page-frame number, page offset, and length). The ATC can contain up to 48 entries. The system indexes the ATC with an AON and offset to produce a physical descriptor directly. This avoids a search through the Memory Hash Table. ATC references start while the protection checks are being made.

A hit in the ATC produces the operand's physical descriptor, which is sent to main memory. Logical Address Translation (LAT) forms the physical descriptor if the ATC misses. Entries are replaced on a least recently used basis. An ATC entry for our example FORTRAN variable is shown in Figure A-18 below.

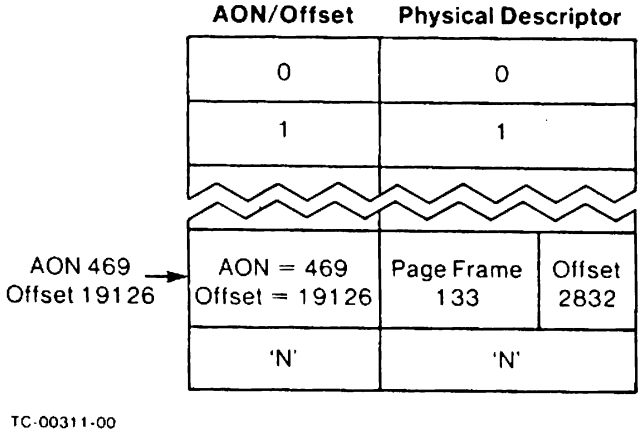


Figure A-18. Address Translation Cache

Protection Cache

The system uses the Protection Cache to bypass an Active Primitive Access Matrix (APAM) reference. The Protection Cache contains access control information for objects that the currently active subject has recently referenced. The system uses the referenced object's Active Object Number to index the cache.

If the subject's rights to the object are in the cache, the rights are checked, and the operand's length information is compared to the length of the object. If the protection check fails, a protection fault signal is sent to the operating system. The system automatically terminates the addressing operation before sending any data to the requestor.

The Protection Cache can hold access information for up to sixteen objects. If the cache misses, the APAM is indexed to produce the subject's access rights to the referenced object. Entries in the cache are replaced on a least recently used basis.

Addressing Summary

This completes our introductory look at the 1000 computer's object-oriented addressing and protection mechanisms. The following illustration summarizes these operations. The illustration shows the system's accelerated paths (through the caches), as well the basic operations that are used when a cache misses.

You can find more details about these operations in the *GALAXY 1000 Theory of Operation* (014-005000).

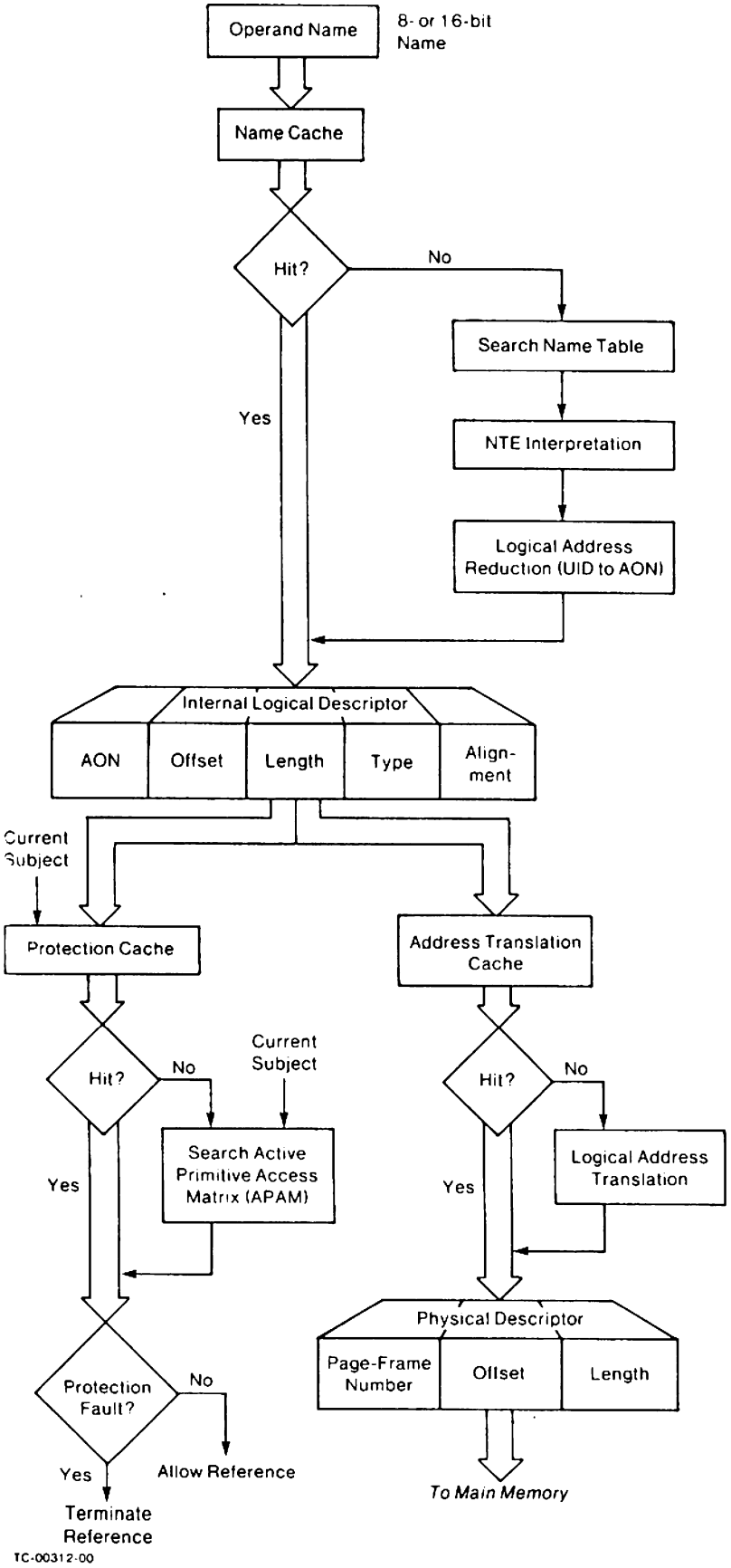


Figure A-19. 1000 Addressing Summary