Virtual Memory, Processes, and Sharing in MULTICS

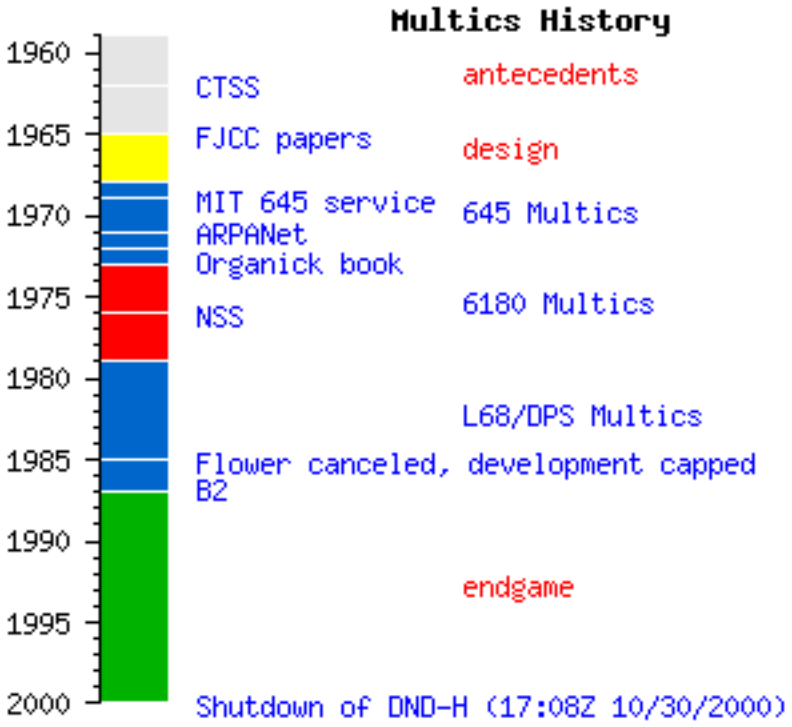
Robert C Daley

- Multics, ARPAnet and CTSS Development Project Manager at MIT
- Currently Chief Architect, Mobile Device Management at HP
- BSME from Tufts, Certificates from MIT and Daniel Webster

Jack B. Dennis

- Emeritus Professor of Computer Science at MIT

1. MULTICS



a. MULTICS: Multiplexed Information and Computing Service

i. Designed for a large community of users
ii. Designed principally for remote terminals

b. Major goals for MULTICS (1965)
   i. Convenient remote terminal use
   ii. Continuous operation analogous to power and telephone services
   iii. Wide range of system configurations (changeable without system or user program reorganization)
   iv. High reliability internal file system
   v. Support for selective information sharing
   vi. Hierarchical structures of information for system administration and decentralization of user activities
   vii. Support for a wide range of applications
   viii. Support for multiple programming environments and human interfaces
   ix. The ability to evolve the system with changes in technology and user aspirations

c. MULTICS features
   i. Memory
      1. 1MB segments, each contains addresses from 0 to 256K words
      2. File system integrated with memory: file access through memory references
      3. Paged memory pioneered by **Atlas** system
         a. Addresses in CPU translated by hardware from virtual to physical address.
         b. Three-level scheme using: main storage, paging device, disk
   ii. Architecture
      1. Supports multiple CPUs sharing same physical memory
      2. Sequential execution
      3. 36-bit Instructions

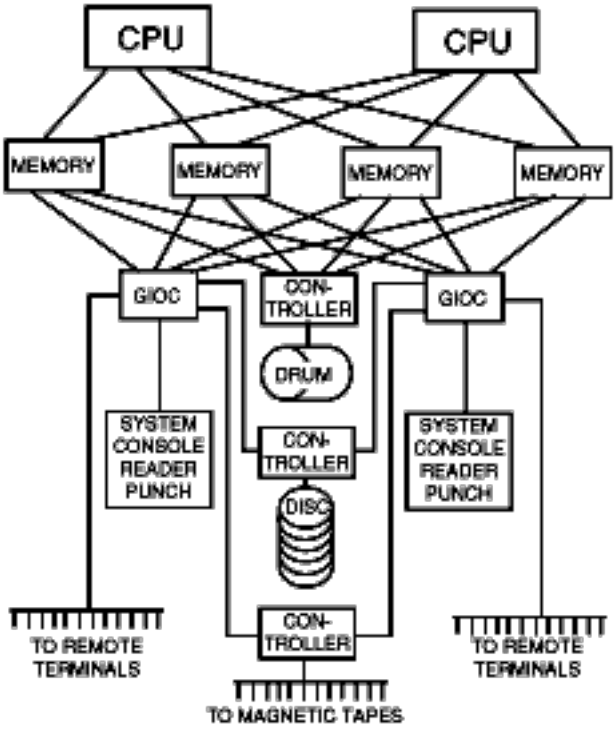
```
 000 000000011111111 112222222 222 333333
 012 345678901234567 890123456 789 012345
+---+---------------+---------+---+------+
|BR |    ADDRESS    | OPCODE  |UIB| TAG  |
+---+---------------+---------+---+------+
```

Field Name	Size	Purpose
BR	3 bits	base register; 0-7
ADDRESS	15 bits	word address; 0-32767 (32KW)
OPCODE	9 bits	instruction opcode
U	1 bit	unused
I	1 bit	interrupt inhibit flag
B	1 bit	0=no base register (GE-635 form) 1=base register (GE-645 form)
TAG	6 bits	index/indirect type

      a. br = segment tag
      b. b = external flag
      c. tag = addressing mode

iii. Flexibility
    1. CPUs, memory, I/O, and disk drives can be added or removed while the system is running
iv. High-level language support
    1. PL/I, BCPL, BASIC, APL, FORTRAN, LISP, SNOBOL, C, COBOL, ALGOL 68, Pascal
    2. Only a small part of the OS implemented in assembly
v. Security
    1. Awarded B2 security rating in 1980s
vi. Database
    1. First commercial relational database product (Multics Relational Data Store (MRDS)) in 1978

d. History
    i. CTSS (Compatible Timesharing System) in 1961 was precursor
        1. MIT Computation Center, IBM 709 (later IBM 7094)
    ii. Project MAC (Multiple Access Computers and Man and Computer)
        1. Developed MULTICS (with ARPA grants, GE, Bell labs)
    iii. General Electric won bid for hardware, GE-645



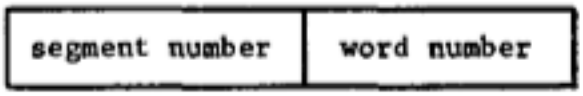
    iv. Phase 1 (December 1967)
        1. Native boot of MULTICS on GE-645
        2. Self hosting
        3. Able to compile itself on MULTICS using bootstrapped compilers by 1968
    v. MULTICS development moved from CTSS to MULTICS in 1969
    vi. Native MULTICS compiler in 1969

vii. Honeywell bought GE in 1970s
    1. Honeywell 6180 (1976)

viii. Last known instance was shut down in 2000
ix. Source code released in 2007
    1. http://web.mit.edu/multics-history/
x.

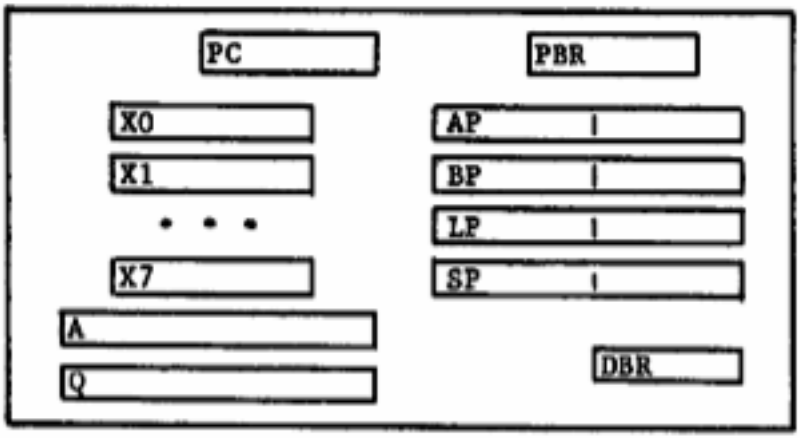
2. Single-Level (Virtual) Memory (this paper)
    a. Objectives addressed
        i. Give the system responsibility for managing distribution of information among levels of storage hierarchy
        ii. Permit a degree of programming generality not previously practical
        iii. Permit sharing of procedures and data among users subject only to proper authorization
    b. Processes
        i. Activity carrying out computation specified by a program
        ii. Each process has its own address space
        iii. Scheduled on the processor by traffic controller
    c. Generalized Address
        i. 2^{14} segments consisting of up to 2^{18} 36-bit words
        ii. Each segment approx. 1MB; address space of approx. 16MB
        iii. GE-645 had 2MiB of memory
        iv. Large address space to avoid necessity of procedure overlays or movement of data within address space
        v. Two types of segments
            1. Procedure
                a. Instructions to be fetched by processor
                b. Non-self-modifying (**Pure**)
            2. Data
                a. Anything not a procedure, including files!
                b. 16MB allows addressing files as part of memory
            3.

vi. Generalized address is location-independent



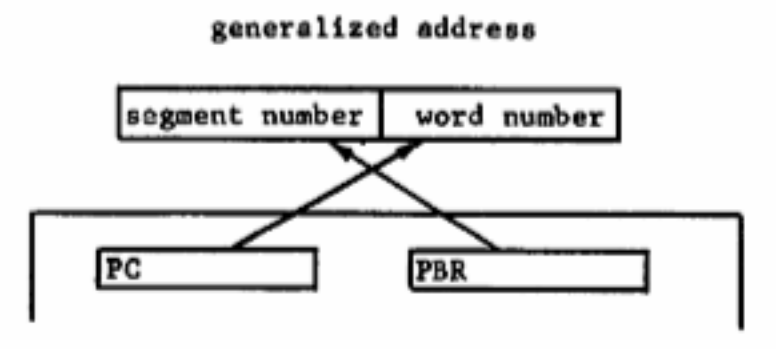
1. Effective reference may be made using generalized address into main memory location
2. If word is not in main memory, supervisor software places address in memory when needed to allow reference by processor

vii. Processor Layout



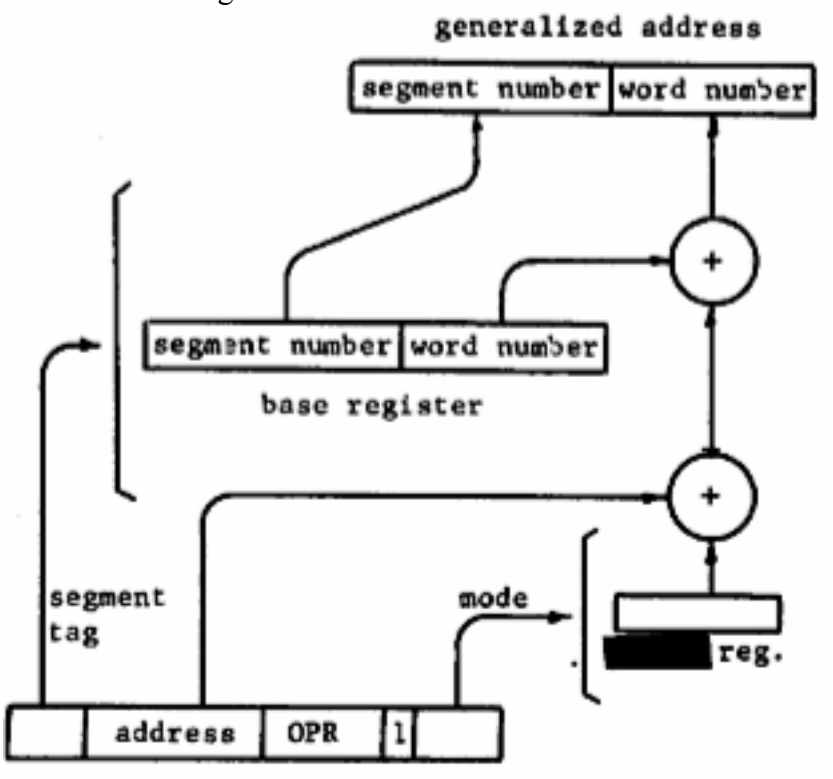
1. A: Accumulator
2. Q: Quotient/Multiplier
3. X0…X7: index registers
4. PBR: Procedure Base Register: Segment number of procedure being executed
5. PC: Program Counter
6. DBR: Descriptor Base Register
    a. Only location-dependent information in the processor
    b. Context switch only requires changing DBR for the new process
7. Base Registers: Complete address
    a. AP: argument pointer
    b. BP: base pointer
    c. LP: linkage pointer
    d. SP: stack pointer

viii. Instruction Fetch Address Generation

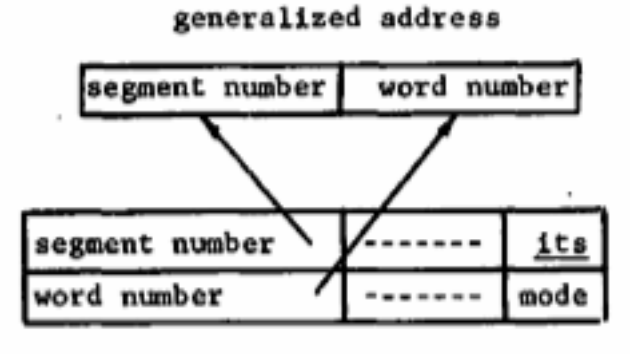


1. Word number from the PC and segment number from PBR

ix. Data Access Address Generation
    1. External Flag OFF
        a. PBR gives segment number
        b. Word number = address field of instruction + selected index register
    2. External Flag ON
        a. Segment tag picks one of 4 Base Registers
        b. BR gives segment number
        c. Word number = address field of instruction + selected index register + word number of BR
        d.



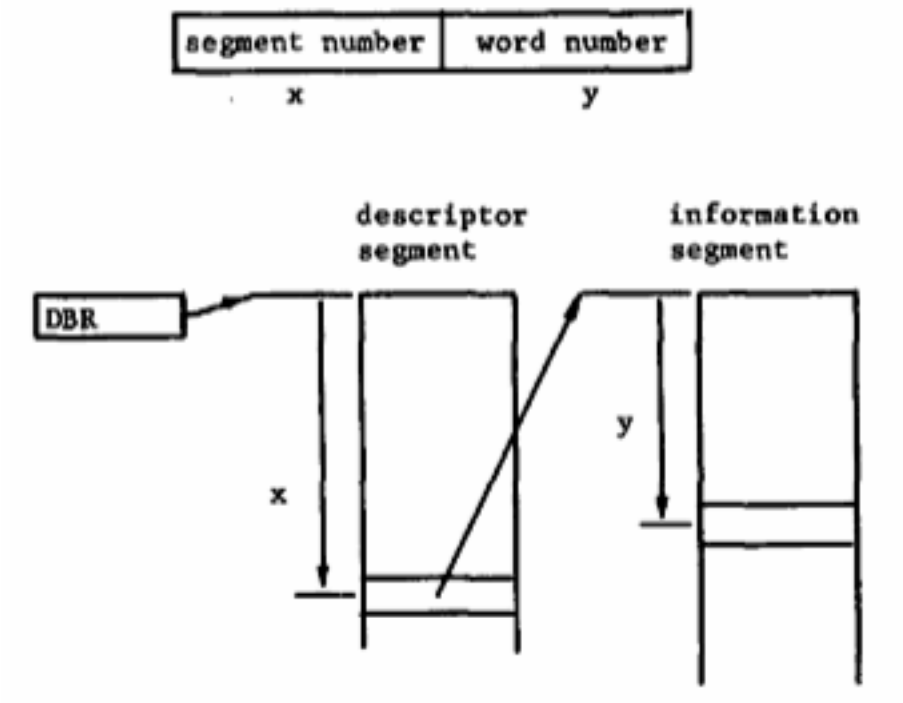
x. Indirect Addressing



1. Instruction may indicate indirect addressing in TAG
2. Two words are fetched and parsed in this manner
    a. If Indirect To Segment (ITS) is set, directly use segment and word numbers

xi. Addressing Using the Generalized Address

1. Two-step hardware table look-up procedure (creating the effective address)



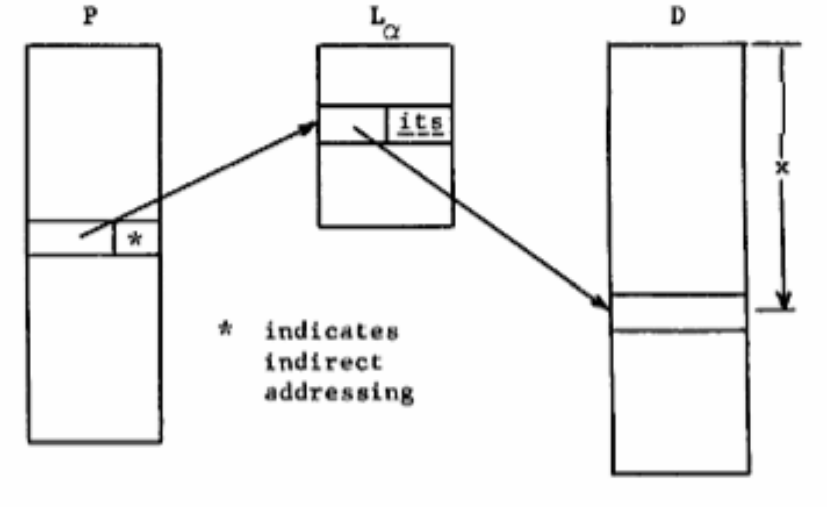
    a. Using the Descriptor Base Register, locates the descriptor segment for the current process
        i. Gives protection of the segment for this process
    b. Descriptor segment will give the address of segment in physical memory
    c. Use word number to index into the segment

xii. Outcomes of Generalized Address

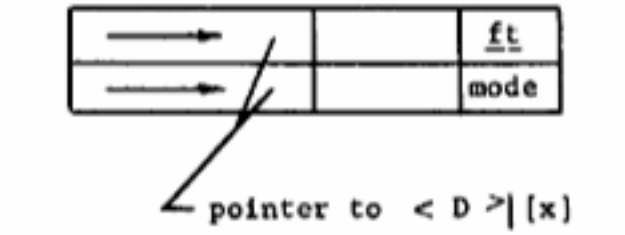
1. The same segment in memory may be identified by different segment numbers in different processors
2. Mapping of generalized addresses to physical locations is transparent to the user
3. Paging will allows non-contiguous physical addresses to be accessed as contiguous generalized addresses
    a. Implemented in a page table in main memory
    b. Pages can be relocated to disk when not in use

3. Intersegment Linking and Addressing
    a. Allowing multiple processes to share procedure (instructions) and data segments
    b. Requirements
        i. Procedure segments must be pure
        ii. Must be possible for a process to call a routine by its symbolic name without having made prior arrangements for its use
            1. Subroutine must have space for data, reference any needed data object, call further routines
        iii. Segments of procedure (instructions) must be invariant to the recompilation of other segments
            1. Any internal addresses that change on recompilation must not appear in any other segment (recompilation will alter the address links)
    c. Making a Segment Known
        i. Given a symbolic name for a segment, a slot in the descriptor segment (lookup table) is allocated for this segment.
        ii. System will correctly fill in the current physical location of the segment into the descriptor segment.
    d. Linking Data
        i. Before a segment is known, it may only be addressed using the symbolic path in the directory structure
        ii. Segment reference name is all a procedure knows, so that must be translated to the path name by directory searching algorithm (what?)
        iii. Once known, set up descriptor segment for indexing into segment (mapping to the address space)
        iv. Program will contain **linkage section** to which external references in the instructions will be made
        v. Every segment will have a symbol table as part of the segment
            1. Table for the system to find a word number in the segment (x) from a symbolic word name ([x]).
            2. In a standard location in all segments

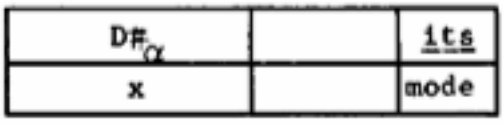
vi. Example: Process a, Program P



1. Wants to address: OPR <D> | [x]
    a. Look up data segment at symbolic location <D>
    b. Look up symbolic address [x] in <D>
2. In process a, we want to use physical address D#a | x without changing the instructions stored in P
3. On first access to a the linked segment, system will trap the process, and the linkage section will be loaded into a segment of the process and an indirect address calculated

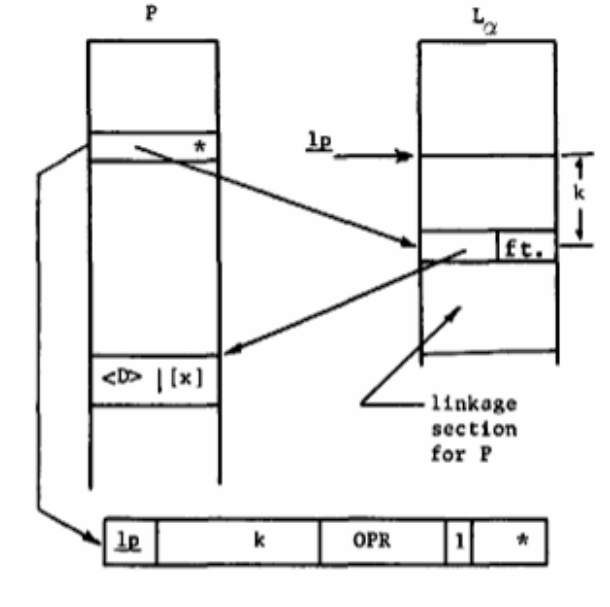


4. Once link established, indirect addressing will provide physical address
    a. La link data entry for the location corresponding to D|x will be filled with the segment number (D#a) and word number (x) for the physical data, and indirect addressing through this entry will give the correct position

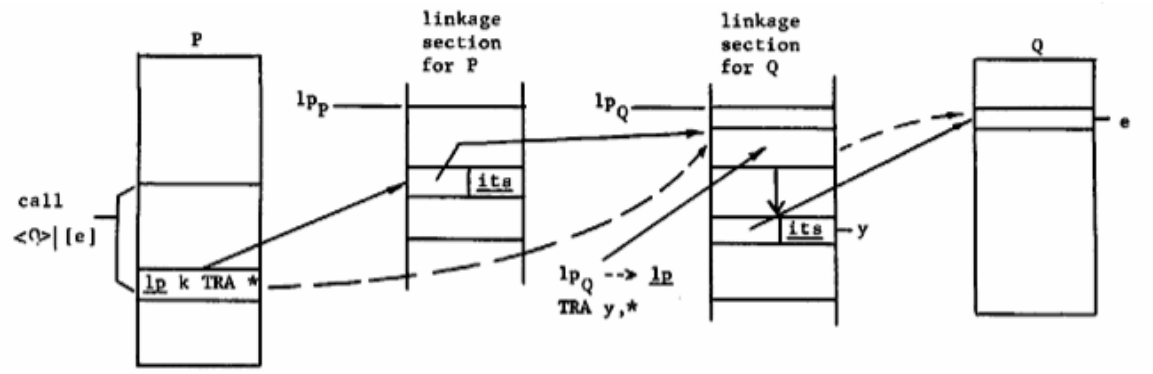


vii. Link pointer LP is the base generalized address for the linkage segment

1. External segment references are coded relative to LP



4. Procedure Calls
    a. Four aspects of subroutine calling
        i. Transmission of arguments
            1. Arguments may be placed on stack or elsewhere
            2. Location of arguments given by AP (argument pointer)
        ii. Arranging for return of control
            1. Machine conditions (PBR, PC, etc) are saved to stack by the caller
        iii. Saving and restoring processor state
        iv. Allocating private storage for called procedure
            1. Frame created and addressed by SP's generalized address (stack pointer)
            2. Frame is released upon return of control
    b. Updating the linkage pointer (LP) on transfer of control
        i. Two additional instructions stored in the linkage section at compile time
            1. Load LP with appropriate value at procedure entry
            2. Transfer control to the called instruction (procedure segment)



P indirectly addresses into the linkage section for Q, executes the instruction to update LP with the linkage pointer for Q, then transfers control to segment Q at entry point e

5. Future Influences
    a. UNIX/LINUX
        i. In 1970, Peter Neumann coined the project name *Unics* (UNiplexed Information and Computing Service) as a pun on *Multics*