IEEE Computer Society Fall Compcon '81

"Research in High-Level Computer Architecture"

John F. Pilat
Data General Corporation/Research Triangle Park

RESEARCH IN HIGH-LEVEL COMPUTER ARCHITECTURE

John F. Pilat

Data General Corporation
Research Triangle Park, NC

INTRODUCTION

This paper presents results from selected areas of research in computer architecture. These efforts are part of an ongoing program at Data General to explore improved foundations for computer system design in the 1980s. Our major goals have been to:

- Provide a very large logical address space for programs.
- Provide for the controlled sharing of information in a distributed, multi-user environment.
- Provide an efficient, cost-effective vehicle for executing programs written in high-level languages.

THE PROGRAM ARCHITECTURE

As we pursued these goals, our research generated a number of architectural ideas. We feel they represent significant advances over conventional designs. Our architectural model has three layers: *data services*, *instruction services*, and *process services*. Interfaces between these layers are very narrow. Changing a layer's internal mechanisms while preserving its interface yields interesting variants that still belong to the architectural family.

Data Services

The data services layer provides the illusion of a large, segmented, bit-addressable virtual memory. This space is composed of up to 2^{80} independent *objects*. Each object can be from zero to 2^{31} bits long. This gives programs a maximum address space of 2^{111} bits (over 10^{32} bytes). These data are totally shareable; there is no restriction to a per-process or per-processor address space. Pointers may denote any bit in this memory.

Objects are protected containers of information. The data services layer also provides basic object protection services. It validates every read, write, or execute request as it is made. A request is made on behalf of a *subject*. This unit of authority is based on user identity, process identity, and domain identity. Whether a subject may access an object in a particular way is controlled by the object's *Access Control List*. This layer also detects any attempt to access past the end of an object.

Instruction Services

The instruction services layer executes instruction streams from *procedure objects*. We have designed this part of the architecture to:

- Execute only one instruction for a typical source statement.
- Interpret only one fixed-length operand reference per source data reference.
- Execute no additional instructions for typical operand references.

Instructions are simple sequences of fixed-size *syllables*. Each instruction begins with an operator syllable. This selects an operation of the *S-language* into which the program was compiled. These are operator sets tailored to the needs of particular high-level languages. They include multiple-address, memory-to-memory formats that match language features such as:

- Assignment: A = B + C;
- Comparison: IF A > B THEN ... ;
- Invocation: CALL PROC (A, B, C);

An operation-specific number of operands follow the operator, one syllable per operand. Interpretation of memory reference operand syllables is fixed across S-languages.

In many cases, the reference syllable suffices for effective address formation. Otherwise, the syllable indexes into a *Name Table* within the current procedure object. References to packed items, array elements, strings, substrings, and based records all use the Name Table.

Process Services

The process services layer manages a program's execution environment. The layer provides various Call operators and a canonical Return. It handles stacked activation records, automatic storage, and by-reference argument passage. It also supports non-local transfers and condition signalling.

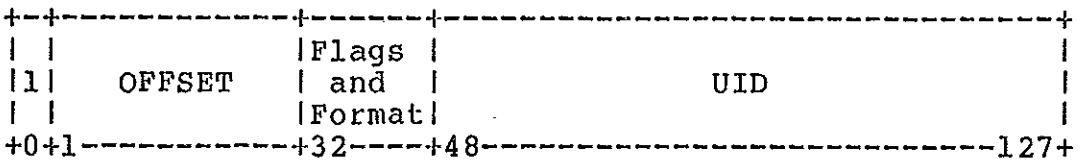
A process can switch procedure objects, Name Tables, S-interpreters, and addressing modes only through call and return. With proper access, a process' subject can also change. It does so as a result of calling a routine in a different *domain*. The domain component of the process' subject changes to reflect the static protection sphere of the called routine. This feature can be used to build secure subsystems and abstract types.

POINTERS

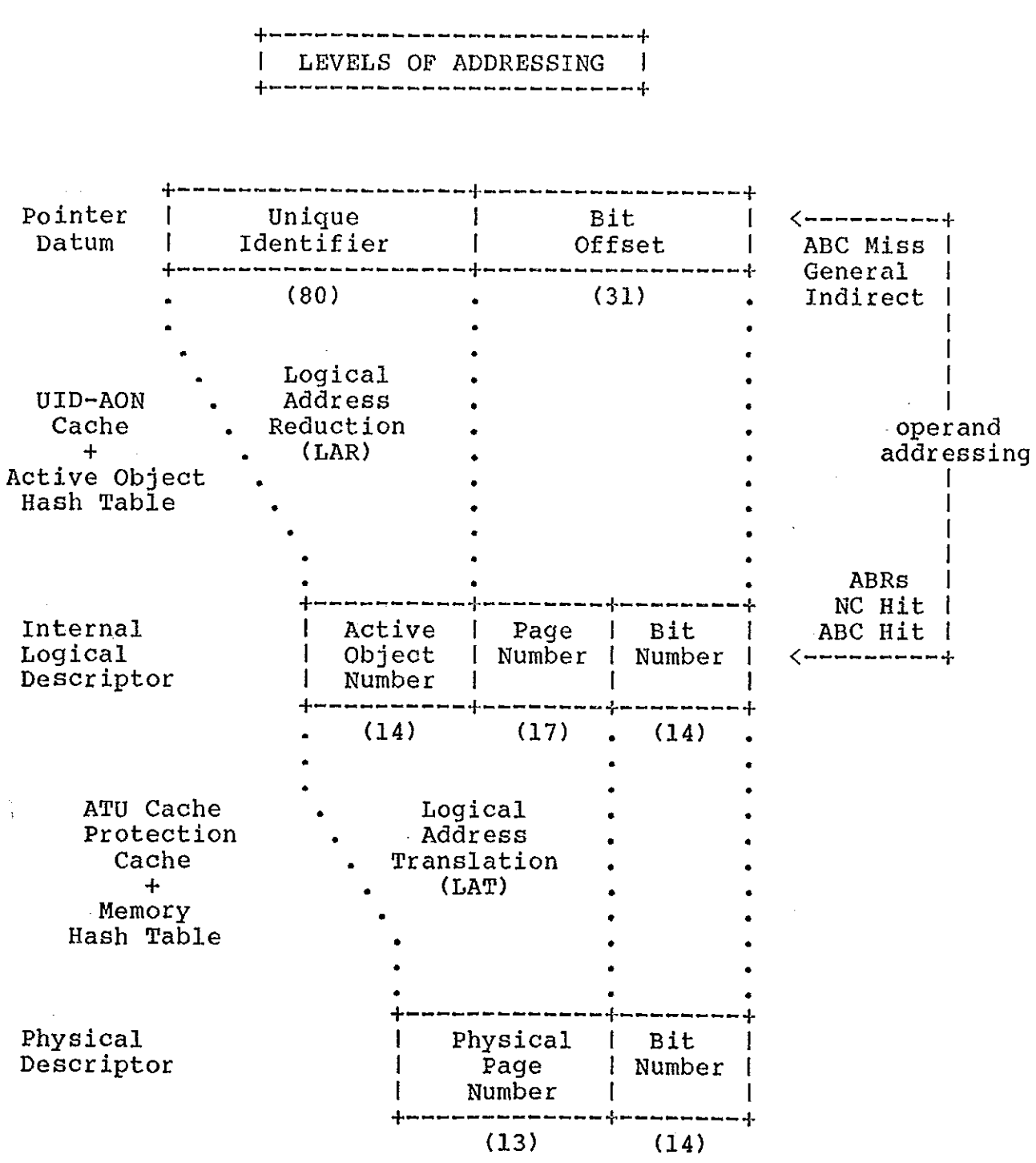
A pointer denotes a bit in the FHP virtual memory.
Dereferencing a pointer selects an object by UID,
and a bit offset within that object.

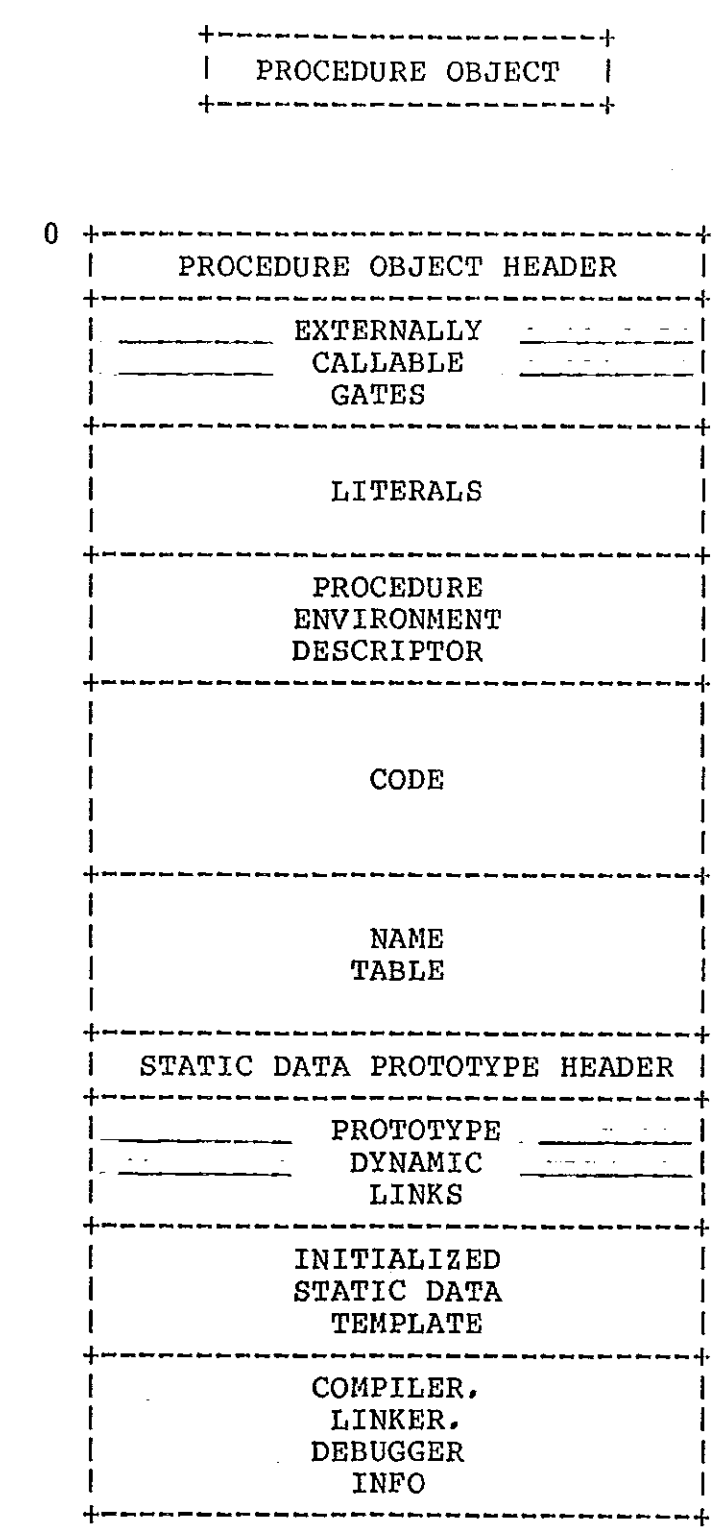
INTRA-OBJECT POINTER
Only the first 32 bits
are significant

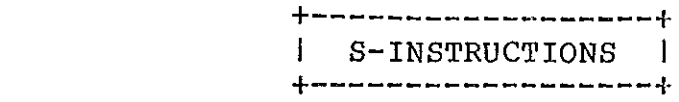
GENERAL POINTER
Can Denote Any Bit
All 128 bits significant



FLAGS / FORMAT FIELD
Controls Interpretation

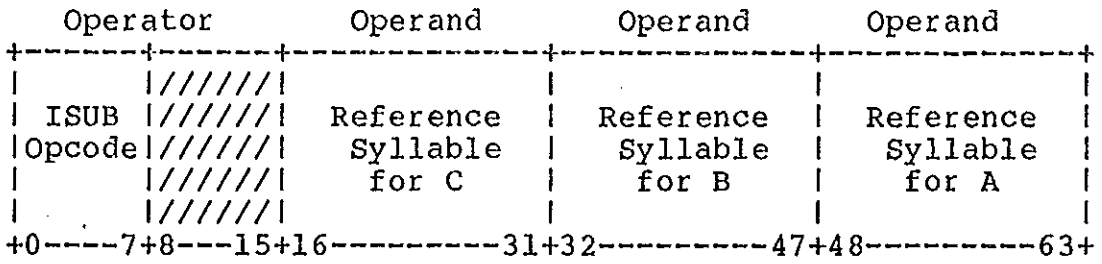

LEVELS OF ADDRESSING
Pointer Datum
Unique Identifier
Bit Offset
ABC Miss General Indirect
(80)
(31)
UID-AON Cache + Active Object Hash Table
Logical Address Reduction (LAR)
operand addressing
ABRs NC Hit ABC Hit
Internal Logical Descriptor
Active Object Number
Page Number
Bit Number
(14)
(17)
(14)
ATU Cache Protection Cache + Memory Hash Table
Logical Address Translation (LAT)
Physical Descriptor
Physical Page Number
Bit Number
(13)
(14)


PROCEDURE OBJECT
0
PROCEDURE OBJECT HEADER
EXTERNALLY
CALLABLE
GATES
LITERALS
PROCEDURE
ENVIRONMENT
DESCRIPTOR
CODE
NAME
TABLE
STATIC DATA PROTOTYPE HEADER
PROTOTYPE
DYNAMIC
LINKS
INITIALIZED
STATIC DATA
TEMPLATE
COMPILER,
LINKER,
DEBUGGER
INFO



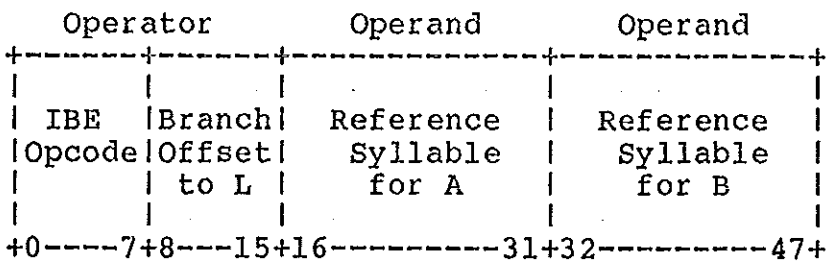
COMPUTATION:

Source Code : A := B - C
Object Code : ISUB C, B, A



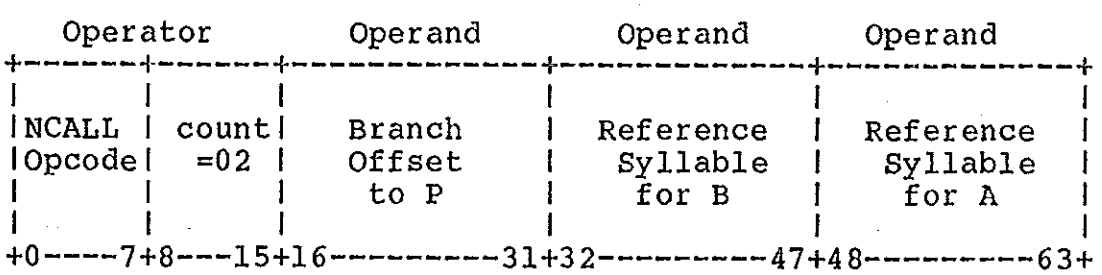
COMPARISON:

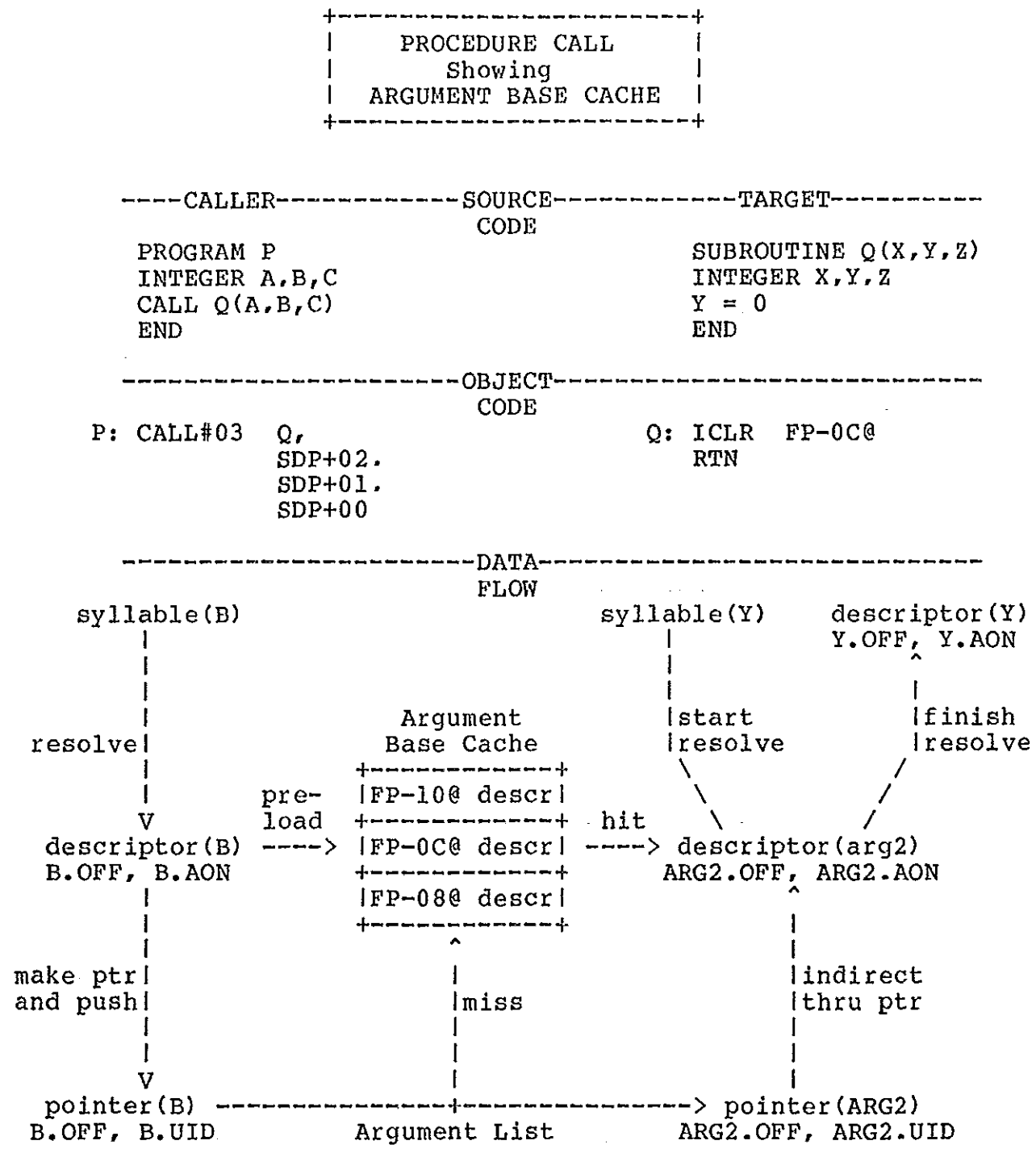
Source Code : IF A = B GO TO L
Object Code : IBE A. B, <L>

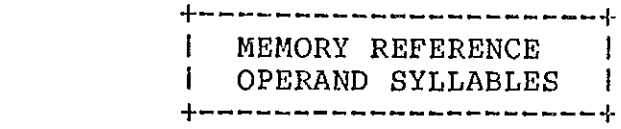


INVOCATION:

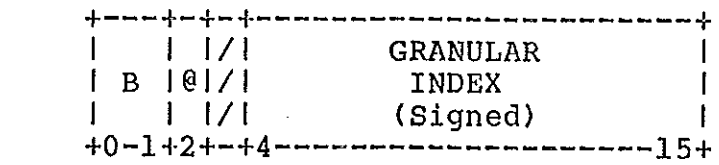
Source Code : CALL P (A. B)
Object Code : NCALL#02 <P>, B, A




PROCEDURE CALL
Showing
ARGUMENT BASE CACHE
CALLER
SOURCE CODE
TARGET
PROGRAM P
INTEGER A,B,C
CALL Q(A,B,C)
END
SUBROUTINE Q(X,Y,Z)
INTEGER X,Y,Z
Y = 0
END
OBJECT CODE
P: CALL#03 Q,
SDP+02.
SDP+01.
SDP+00
Q: ICLR FP-0C@
RTN
DATA FLOW
syllable(B)
resolve
descriptor(B)
B.OFF, B.AON
make ptr
and push
pointer(B)
B.OFF, B.UID
Argument
Base Cache
pre-
load
FP-10@ descr
FP-0C@ descr
FP-08@ descr
miss
Argument List
syllable(Y)
start
resolve
hit
descriptor(arg2)
ARG2.OFF, ARG2.AON
descriptor(Y)
Y.OFF, Y.AON
finish
resolve
indirect
thru ptr
pointer(ARG2)
ARG2.OFF, ARG2.UID



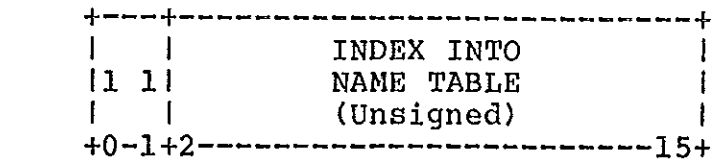
GRANULAR REFERENCES:



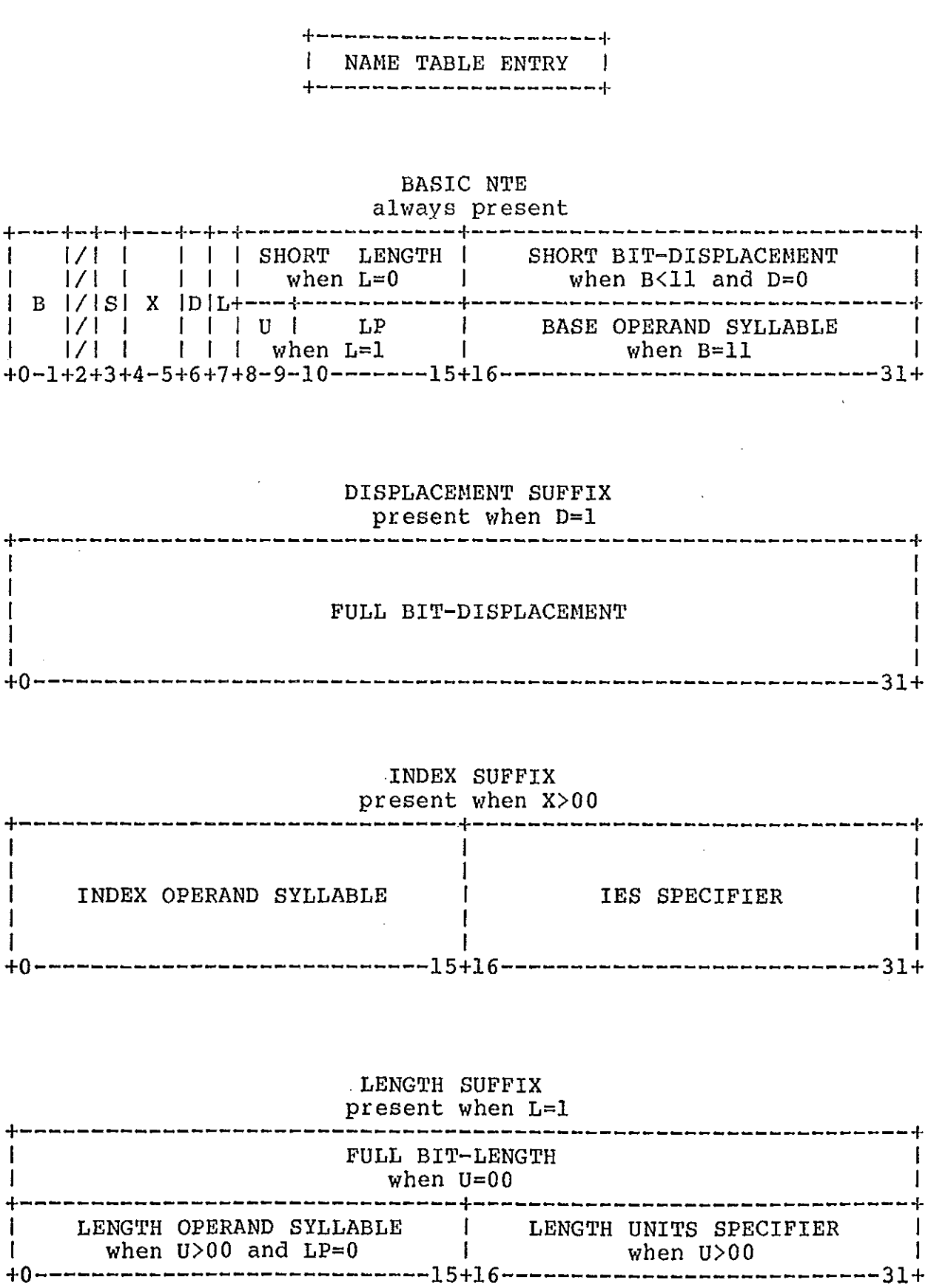
Effective address calculation:

1) Use the ABR selected by B.
2) Add 32 * INDEX to the offset portion.
3) If indirect, use pointer found there.
4) Supply a default length of 32 bits.
5) Supply a default type of Signed.

NAMED REFERENCES:



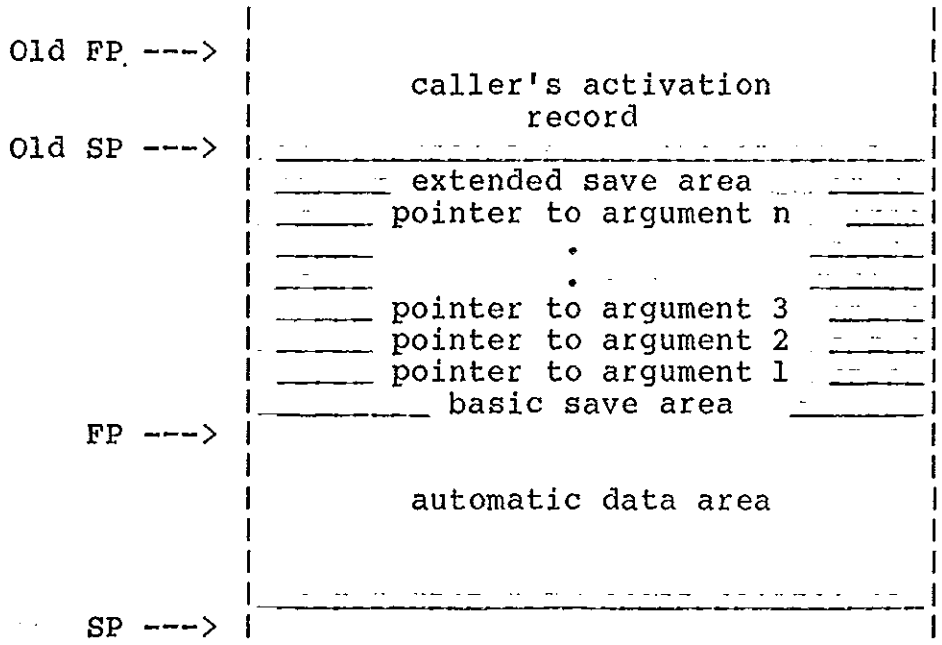
The data item is described by an entry in the current Name Table, located at offset 32 * INDEX.


NAME TABLE ENTRY
BASIC NTE
always present
B
S
X
D
L
SHORT LENGTH
when L=0
SHORT BIT-DISPLACEMENT
when B<11 and D=0
U
LP
when L=1
BASE OPERAND SYLLABLE
when B=11
0 1 2 3 4 5 6 7 8 9 10 15 16 31
DISPLACEMENT SUFFIX
present when D=1
FULL BIT-DISPLACEMENT
0 31
INDEX SUFFIX
present when X>00
INDEX OPERAND SYLLABLE
IES SPECIFIER
0 15 16 31
LENGTH SUFFIX
present when L=1
FULL BIT-LENGTH
when U=00
LENGTH OPERAND SYLLABLE
when U>00 and LP=0
LENGTH UNITS SPECIFIER
when U>00
0 15 16 31

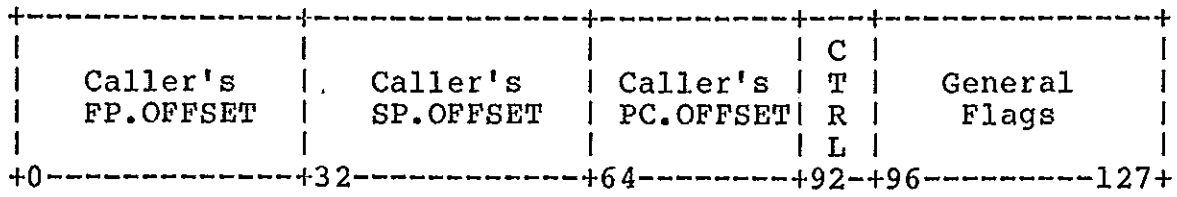
* COMPATIBLE WITH GENERAL CALL AND UNIVERSAL RETURN

* WRITES 96 BITS (NO ARGUMENTS) ONTO PROCESS' STACK

* READS 48 BITS (NO ARGUMENTS) FROM THE CODE STREAM

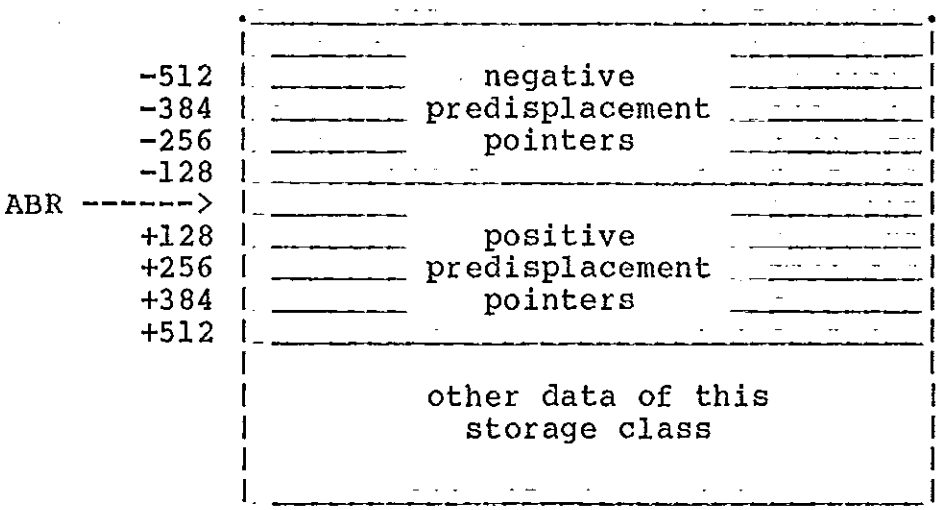


BASIC SAVE AREA
Neighborhood Call Case



All operand addressing begins with one of three Architectural Base Registers. These denote exactly the Storage Classes required by high-level languages.

ABRs contain full memory addresses (UID,OFFSET). However. they are kept internally as Logical descriptors (AON.OFFSET).



Architectural Base Register	Data Area (Direct)	Linkage Area (Indirect)
00 Frame Pointer	AUTOMATIC	PARAMETER
01 Static Data Pointer	INTERNAL STATIC	EXTERNAL STATIC
10 Procedure Base Pointer	CONSTANT	////////////////

```
 1 |
 2 |       MODULE example;
   |         EXPORTS bubblesort;
 4 |
 5 |         PROCEDURE bubblesort (
 6 |           READ WRITE array : ARRAY [0..999] OF INTEGER,
 7 |           READ ONLY  limit : INTEGER );
 8 |
 9 |           VARIABLE
10 |             temp    :  INTEGER,
11 |             swapped :  BOOLEAN BOUND 32;
12 |
13 |           FOR j OVER { 0..limit-2 } DECREASING REPEAT
14 |
15 |             swapped := FALSE;
16 |
17 |             FOR i OVER {0..j} INCREASING REPEAT
18 |
19 |               IF array[i] > array[i+1] THEN
20 |                  temp       := array[i];
21 |                  array[i]   := array[i+1];
22 |                  array[i+1] := temp;
23 |                  swapped    := TRUE;
24 |               END IF;
25 |
26 |             END FOR;
27 |
28 |             WHEN NOT swapped RETURN;
   |
30 |           END FOR;
31 |           RETURN;
32 |
33 |         END PROCEDURE bubblesort;
34 |
35 |       END MODULE example;
36 |
```

/STAT

```
Routine: example\bubblesort      Initial frame size = 00000100

   PC     Object Code              Instructions

13 000290  D5 00 8001 PBP+0020     ISUB     2.
                2FF4 FP-0180@               limit.
                0003 FP+0060                j

   0002D0  E5 1E 0003 FP+0060     IBLZ     j
                                           <3>

15 0002F0  C4 00 0001 FP+0020    0:  ICLR     swapped

17 000310  C4 00 0004 FP+0080     ICLR     i

   000330  CD 00 0003 FP+0060     IMOV     j.
                0002 FP+0040                def.05D80065

19 000360  E3 0E C000            1:  IBLE     array[i],
                C002                        array[i+1]
                                            <2>

20 000390  CD 00 C000             IMOV     array[i],
                0000 FP+0000                temp

21 0003C0  CD 00 C002             IMOV     array[i+1],
                C000                        array[i]

22 0003F0  CD 00 0000 FP+0000     IMOV     temp,
                C002                        array[i+1]

23 000420  D4 00 0001 FP+0020     ISETONE  swapped

26 000440  DB F2 0004 FP+0080    2:  ILPUP    i,
                0002 FP+0040                def.05D80065
                                            <1>

28 000470  E8 04 0001 FP+0020     IBZ      swapped
                                           <3>

30 000490  DA E6 0003 FP+0060     ILPDNZ   j
                                           <0>

33 0004B0  03 00                 3:  RTN
```

```
Name Table at offset 00000D80

NT#   Base  Displacement  Length   FM  --Flags---    Index     IES

#0    FP-0100      +0000  20        s  B.@ . .  I2  FP+0080   +0005
#2    FP-0100  +00000020  20        s  B.@ D .  I2  FP+0080   +0005

Flags:   D = Long Displacement
         B# = Base is a Name    B@ = Base Indirect
         L = Long Length        L# = Length is a Granny
         I = Indexed            I# = IES is a Granny   I2 = IES is Log2
FM:      z = zero fill          s = sign fill
```

SYSTEM BENCHMARK SUITE

	MV/8000 PL/I	FHP SPRINT SPL	
		2300	2400
Bubblesort	1.01	0.48	0.47
Topsort	2.87	2.84	~ 1.91
Squeeze	2.32	0.31	0.31
Ucify	3.35	3.39	~ 1.90
Ackermann	0.46	0.41	0.28
Total Time (seconds)	10.01	7.43	4.87
Instructions	4040	1824	
Name Table		740	
Total Size (bytes)	4040	2564	

BOTTLENECKS IN HIGH-END PERFORMANCE

SIZE OF I-STREAM

NUMBER OF SYLLABLES IN I-STREAM

SERIAL DEPENDENCIES IN I-STREAM

FIXED NUMBER OF ACCELERATORS

HIGH CONTROL OVERHEAD

INEFFICIENT USE OF MULTIPLE PROCESSORS

INEFFICIENT USE OF LARGE MEMORY SYSTEMS

TJ 8/81

FHP PROCEDURE CODE IS EXTRAORDINARILY DENSE

PROCEDURE CODE SIZE (KBYTES)

BENCHMARK NAME	EAGLE	VAX	FHP
WHETSTONE	2.6	1.6	1.3
SPL/PL1 SUITE	4.0	-	3.0

FHP CODE SYLLABLES ARE UNIFORM IN SIZE

* ALL I-STREAM SYLLABLES ARE 16 BITS IN LENGTH

    * OPERATION SPECIFIERS CONSIST OF 8-BIT PRIMARY OPCODE CATENATED WITH 8-BIT MODIFIER. MODIFIER MAY BE SECONDARY OPCODE, LITERAL, ARGUMENT COUNT, ETC.

    * OPERAND SPECIFIERS CONSIST OF SELF DESCRIBING ADDRESS INFORMATION. MOST ADDRESSES CAN BE GENERATED USING INFO IN I-STREAM. MORE COMPLEX ADDRESSES USE OPERAND SPECIFIER TO INDIRECT THROUGH WRITE ONLY "NAME TABLE".

* ALL SYLLABLES ON 16 BIT BOUNDARIES. NO CROSS WORD INSTRUCTION SYLLABLES WHICH SLOW DOWN PARSING.

* FACILITATES VERY HIGH SPEED ADDRESS FORMATION. THIS ALLOWS A FETCH UNIT TO BE BUILT WHICH IS CAPABLE OF BALANCING MODERN E-BOX TECHNOLOGY.

* CONTRASTS WITH VAX, EAGLE, iAPX 432

TJ 8/81

IMPLICIT ACCELERATORS

* WITH THE EXCEPTION OF A SINGLE ACCUMULATOR. FHP IS ARCHITECTURALLY A MEMORY BASED MACHINE. OPERAND SPECIFIER SYLLABLES ARE DYNAMICALLY RESOLVED TO POINT TO MEMORY LOCATIONS.

* MOST OPERAND SPECIFIERS ARE UNIQUE WITHIN THEIR USAGE DOMAIN. THIS FEATURE ALLOWS OPERAND SPECIFIER SYLLABLES TO BE USED AS ASSOCIATIVE KEYS TO ACCELERATE DATA ACCESSES.

* RESOLVED MEMORY ADDRESSES MAY BE ENCACHED USING OPERAND SPECIFIER SYLLABLE AS ASSOCIATIVE KEY. (OPERAND SPECIFIERS ARE USED AS VIRTUAL REGISTER ADDRESS IN REGISTER SET CONTAINING ADDRESSES.)

* DATA MAY BE ENCACHED USING OPERAND SPECIFIER SYLLABLE AS ASSOCIATIVE KEY. (OPERAND SPECIFIERS ARE USED AS VIRTUAL REGISTER ADDRESSES IN REGISTER SET CONTAINING DATA.)

* PARTIALLY RESOLVED MEMORY ADDRESSES MAY BE ENCACHED USING OPERAND SPECIFIER SYLLABLE AS ASSOCIATIVE KEY. (OPERAND SPECIFIERS ARE USED AS VIRTUAL ALGORITHM SPECIFIERS WHICH SPECIFY ADDRESSES AND DATA IN REGISTER FILE AND MICROCODE ROUTINE TO DYNAMICALLY COMPLETE ADDRESS CALCULATION.)

* ACCELERATORS ARE TRANSPARENT TO SOFTWARE. THEY MAY BE IMPLEMENTED ON A MACHINE DEPENDENT BASIS TO ACHIEVE DESIRED COST/PERFORMANCE. SOFTWARE MAY BE OPTIMIZED FOR A SPECIFIC ACCELERATOR SET. BUT WILL RUN UNALTERED ON OTHER MACHINES.

MINIMUM CONTROL OVERHEAD

* ONE INSTRUCTION PASSES ARGUMENTS, PUSHES STATE BLOCK, AND TRANSFERS CONTROL.

* COST OF CONTROL TRANSFERS DEPEND ON RELATIVE POSITION OF TARGET.

* INTRA-PO CALL IS VERY FAST. RETURN PC, FRAME POINTER, AND STACK POINTER OFFSETS MUST BE SAVED (96 BITS TOTAL). IN STATICALLY BOUND PROGRAMS, ALMOST ALL CALLS ARE INTRA-PO.

* ONLY CROSS DOMAIN CALLS MUST PAY FOR ARCHITECTURAL SECURITY MECHANISMS. TO PROVIDE SIMILAR SECURITY OTHER SYSTEMS MUST RESORT TO SOFTWARE INTERPRETATION.

TJ 8/81

IMPLICIT MULTIPROCESSING

* PROCESS STATE OF RUNNABLE PROCESS IS CONTAINED IN "VIRTUAL PROCESSOR". NUMBER OF VIRTUAL PROCESSORS IS SYSTEM PARAMETER.

* UNI-PROCESSOR TIME MULTIPLEXES PHYSICAL PROCESSOR AMONG VIRTUAL PROCESSORS. MULTI-PROCESSOR SYSTEM CAN RUN MULTIPLE VIRTUAL PROCESSORS CONCURRENTLY.

* THIS ABSTRACTION ALLOWS EFFICIENT USE OF ADDITIONAL PROCESSORS. WITHOUT SYSTEM REDESIGN.

TT 8/01

FILE SPACE (UID) ADDRESSING

* DIRECT VIRTUAL ACCESS TO FILES MAKES FILE I/O BUFFERING UNNECESSARY. ALLOWS SYSTEM TO USE ARBITRARY QUANTITIES OF MEMORY.

* LOW LEVEL PRESENCE OF FILE ADDRESSING ALLOWS HARDWARE AND MICROCODE ACCELERATION OF FILE ACCESS ON A MODEL DEPENDENT BASIS.

TJ 8/81