Oct 1957

PROJECT STRETCH

Examples of the Half-Word Index Scheme in Action

1. Chain Indexing
2. Non-Associatively Indexed Indirect Addressing
3. Associatively Indexed Indirect Addressing
4. Merging

E. F. Codd
October 7, 1957

DATE OCT 2/57
DRAWN E F C
CHECKED

1. CHAIN INDEXING

Specific Example

A list is stored in memory to indicate which areas in memory are currently available. A typical entry in this list appears as follows:

LOCATION	CONTENTS	COMMENTS
R_i	R_{i+1}	Reset address
$R_i + 1$	V_i	Starting address for i^{th} area
$R_i + 2$	C_i	No. of full words available in i^{th} area

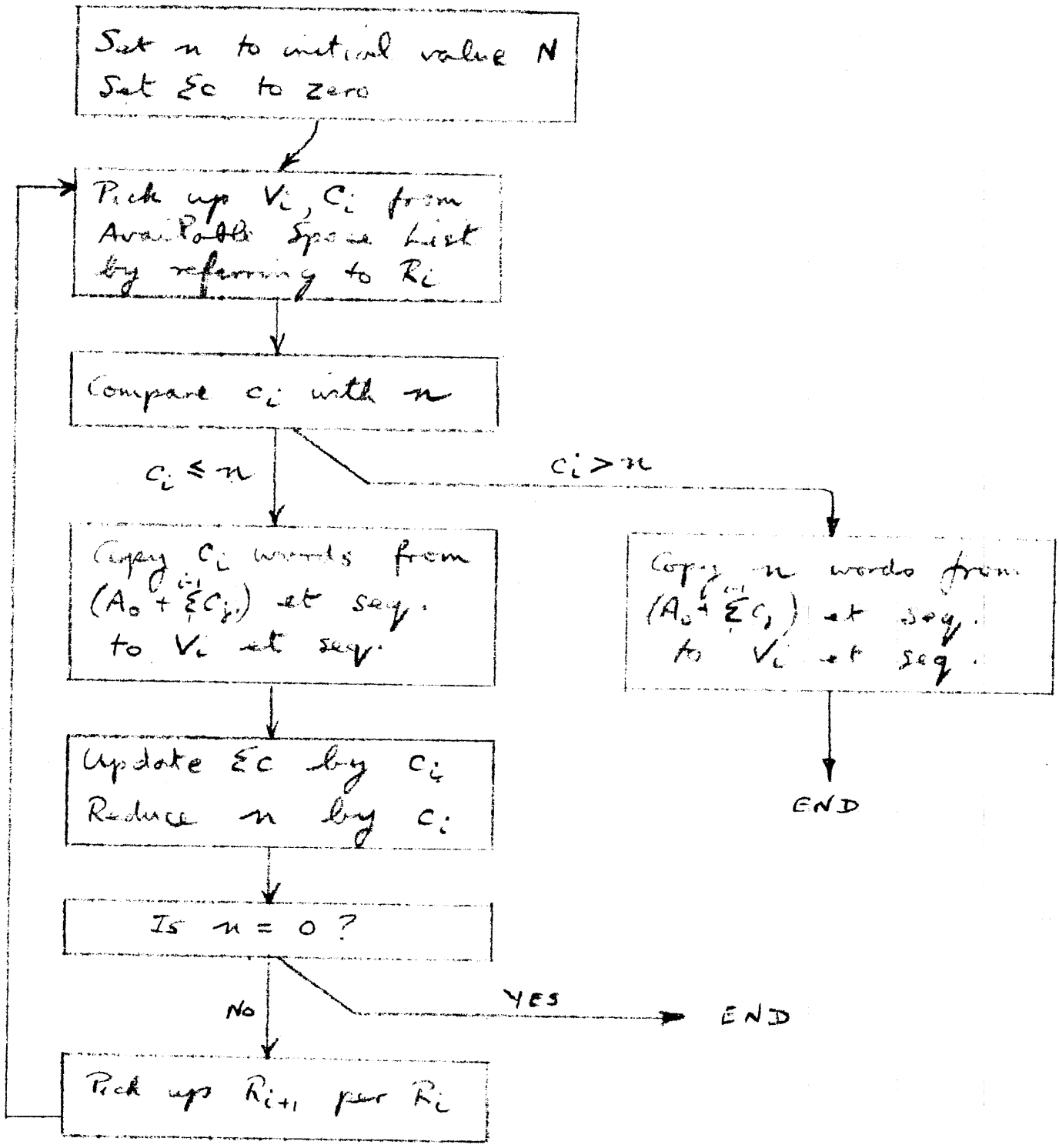
The reset address chains successive entries together. The task to be performed consists of copying N full words from a set of consecutive locations beginning at A into the first N available locations designated by the Available Space List. The starting address of the Available Space List is given in a known fixed location, say index register 2.

The following use is made of index registers during execution of the program:

LOCATION	CONTENTS	
2	R_i	
3	n	No. of words left to be copied
4	V_i	
5	C_i	
6	Σc	No. of words which have been copied

DATE OCT 2 / 57
DRAWN EFC
CHECKED

CHAIN INDEXING (cont'd)



IBM

DATE 12/57
DRAWN EFC
CHECKED

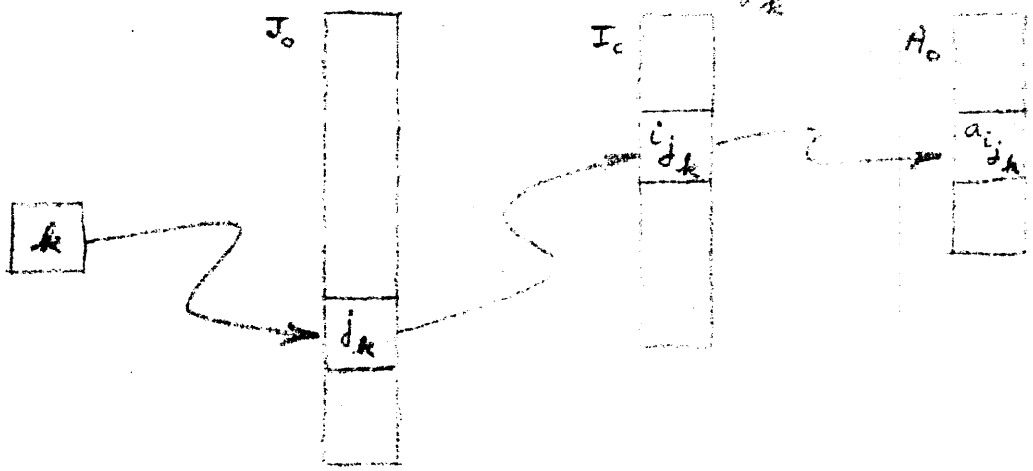
LOCATION	OPERATION	TAG	ADDRESS	
	Load Index 3		L(N)	$c(3) = N = n$
	Load Index 6		0	$c(6) = 0$
λ →	Load Index 4	2	1	$c(4) = c(c(2)+1) = V_i$
	Load Index 5	2	2	$c(5) = c(c(2)+2) = C_i$
	Comp. Index 5		3	C_i versus n
	Branch Index High		θ	Go to θ if $C_i > n$
	Transmit from	6	A_0	Copy C_i words to fill
	to	4	0	ith available block
	length	5	—	
	Add Index 6		5	$c(6) = c(6) + c(5)$
	Sub Index 3		5	$c(3) = c(3) - c(5) = n$
	Branch Index Zero		ϕ	Go to ϕ if $n = 0$
	Load Index 2	2	0	$c(2) = c(c(2)) = R_{i+1}$
	Branch		λ	Go to λ
θ	Transmit from	6	A_0	Copy n words into ith
	to	4	0	available block to
	length	3	—	complete all copying
ϕ	END			

DATE Oct 3/57
DRAWN EFC
CHECKED

2. Non-Associatively Indexed Indirect Addressing

Statement of Problem

A_0, I_0, J_0 are given base addresses
Find $a_{i_{j_k}}$ from index k



$c(k + J_0) = j_k$ Step 1: Use k to pick up j_k

$c(j_k + I_0) = i_{j_k}$ Step 2: " j_k " " i_{j_k}

$c(i_{j_k} + A_0) = a_{i_{j_k}}$ Step 3: " i_{j_k} " " $a_{i_{j_k}}$

Suppose the given value of k is located in index register 2

OP	TAG	ADDR	REMARKS
Load Index 2	2	J_0	$c(2) = j_k$
Load Index 2	2	I_0	$c(2) = i_{j_k}$
Load Index 2	2	A_0	$c(2) = a_{i_{j_k}}$

DATE Oct 3/57
DRAWN EFC
CHECKED

3. Associatively-Indexed Indirect Addressing

Statement of Problem

Given an operand address a_1 and an index address b_1, combine them to form an effective address $E_2 = a_1 + c(b_1)$. Consult E_2 and some related address (for example, $E_2 + 1$) to obtain a second operand address a_2 and a second index address b_2. Combine these to form a new effective address E_3, and so on. This process is continued until <u>either</u> (i) an operand address is picked up and its interrupt tag bit = 1

<u>or</u> (ii) a prescribed number of levels has been executed.

Single Indexing and Multiple Indexing

The above statement is framed in terms of single indexing at each level. Actually, many indices may be needed to obtain the next effective address:

$$E_{k+1} = a_k + c(b_k) + c(c_k) + c(d_k) + \cdots$$

Example 1 demonstrates a technique for executing the k^{th} level of associatively-indexed indirect addressing with only one index at each level. Example 2 deals with multiple indexing at each level.

DATE Oct 3/57
DRAWN E F C
CHECKED

Example 1

INITIAL STORAGE LAYOUT

LOCATION	CONTENTS	LOCATION	CONTENTS	REMARKS
L_k	a_k	2	$a_{k-1} + c(b_{k-1})$	$= L_k$
$L_k + 1$	b_k	3	irrelevant	

PROGRAM →

OP	TAG	ADDR	REMARKS
Load Index 3	2	1	$c(3) = b_k$
" " 2	2	0	$c(2) = a_k$
Add Index 2	–	3	$c(2) = a_k + c(b_k) = L_{k+1}$

Example 2

$c(L_k) = a_k$; $c(L_k + 1) = b_k$; $c(L_k + 2) = c_k$; $c(L_k + 3) = d_k$

$c(2) = L_k = a_{k-1} + c(b_{k-1}) + c(c_{k-1}) + c(d_{k-1})$.

OP	TAG	ADDR	REMARKS
Load Index 3	2	0	$c(3) = a_k$
Load Index 4	2	1	$c(4) = b_k$
Add Index 3	–	4	$c(3) = a_k + c(b_k)$
Load Index 4	2	2	$c(4) = c_k$
Add Index 3	–	4	$c(3) = a_k + c(b_k) + c(c_k)$
Load Index 4	2	3	$c(4) = d_k$
Add Index 3	–	4	$c(3) = a_k + c(b_k) + c(c_k) + c(d_k)$
Load Index 2	–	3	$c(2) = L_{k+1}$

Note: The instruction "Load Index Indirect" is now available, and this takes care of the singly indexed case automatically.

DATE OCT 3/57
DRAWN EFC
CHECKED

4. MERGING

Statement of Problem

Scattered through memory are two strings of records. Each string is composed of N of these records. Elsewhere in memory there are two lists of addresses: List I corresponds to String I; List II to String II.

	LOCATION	CONTENTS		LOCATION	CONTENTS	
	p	m_0		q	n_0	
	p+1	m_1		q+1	n_1	
LIST I	p+2	m_2		q+2	n_2	LIST II
	p+3	m_3		q+3	n_3	
	⋮	⋮		⋮	⋮	
	p+j	m_j		q+k	n_k	
	⋮	⋮		⋮	⋮	

Each address m_j represents the location of a key field in one of the records in String I. All keys are assumed to have the same field length and byte size. Addresses in List I are ordered as follows: m_0 is the address of the lowest key; m_1 is the address of the next lowest key; and so on. Similar remarks apply to List II and String II.

The two strings are now to be merged in the sense that a new list of addresses List III is to be produced representing the order which would pertain if all records were regarded as belonging to one string (String III)

DATE Oct 3/57
DRAWN EFC
CHECKED

MERGING (continued)

For example List III might appear as follows:

LOCATION	CONTENTS
r	n_0
r+1	n_1
r+2	n_0
r+3	n_2
r+4	n_3
r+5	n_4
r+6	n_1
r+7	n_2
⋮	⋮

Note:

$c(n_j)$ = Key in j^{th} record in String I

$c(n_k)$ = " " k^{th} " " " II

This ordering implies that:

Key #3 II ≤ Key #4 II ≤ Key #1 I

Solution of Problem

The process is started by consulting the top members of Lists I + II: from location p, the address n_0 is obtained; from location q, the address n_0 is obtained. Now, each of these addresses is consulted to obtain the corresponding keys. These keys are compared in the main accumulator. Let us say that the key from n_0 is the smaller of the two keys. Then, n_0 is recorded at the top of List III. It is now necessary to advance the "pick-up address" for List II and the "putaway address" for List III in preparation for the next comparison. Further, List II must be consulted to obtain n_1. The process continues in a similar fashion until all 2N addresses

DATE Oct 3/57
DRAWN EFC
CHECKED

have been recorded in List II.

USE OF INDEX REGISTERS

LOCATION	CONTENTS	REMARKS
2	COUNT	Initially set to 2N
3	m_j	Addr of key of j^{th} record Group I
4	n_k	" " " k^{th} " " II
13	$p+j$	Pickup Address for List I
14	$q+k$	" " " " II
15	$r+l$	Putaway Address for List III

PROGRAM Assume $C(2) = 2N$, $C(13) = p$ $C(14) = q$ and $C(15) = r$.

LOCATION	OP	TAG	ADDR	REMARKS
	Load Index 3	13	0	$C(3) = C^2(13) = m_0$
	Load Index 4	14	0	$C(4) = C^2(14) = n_0$
μ →	VFL Load	3	0	$C(Acc) = C^2(3) =$ Key # j I
	VFL Compare	4	0	versus $C^2(4) =$ Key # k II
	Branch Data High		λ	Go to λ if $C^2(3) > C^2(4)$
	Store Index 3	15	0	Putaway m_j in List III
	Add Index 13 ↺		1	Step up the List I pick up
	Load Index 3	13	0	Pickup next entry from I
	Add Index 15 ↺		1	Step up the List III putaway
	Sub Index 2 ↺		1	Reduce count by 1
	Branch Index NZ	(2)	μ	Continue if count $\neq 0$
	END			
→ λ	Store Index 4	15	0	Putaway n_k in List III
	Add Index 14 ↺		1	Step up List II pick up
	Load Index 4	14	0	Pickup next entry from II
	Branch		ϕ	Go to ϕ

Note: This program is based on a technique suggested by Dr. G. Blaauw.

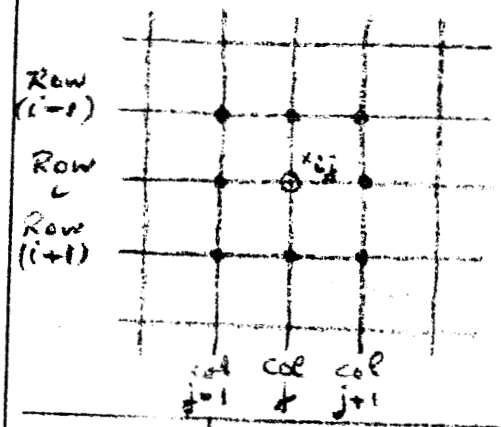
IBM

DATE Oct 3/57
DRAWN EFC
CHECKED

8 NEIGHBOR SUM

$1 \le i \le I-2$

$1 \le j \le J-2$



LOCATION	CONTENTS	
3	I + J - 2	
4	I - 2	
5	Count	
6	Value	
7	Name (initially 19)	
21	I(i-1) + j-1	VALUE
22	3	COUNT
23	I i + j-1	VALUE
24	1	COUNT
25	I i + j+1	VALUE
26	1	COUNT
27	I(i+1) + j-1	VALUE
28	3	COUNT
29		

LOCATION	OPERATION	TAG	ADDR	
H	Load Index 7 II		19	C(7) = 19
	Load		0	C(Acc) = 0
λ	Load Name Index 6	7	2	Name in 7, Value in 6
	Load Index 5	7	1	Count in 5
μ	(Add)	6	A_0	Contribute to Neighbor Sum
	Add Index 6 II		1	C(6) = C(6) + 1
	Count Branch N2 5		μ	C(5) = C(5) - 1
	Comp Index 7 II		29	
	Branch Index Low		λ	
	Store	4	A_0	Store the Neighbor Sum
	Add Index 4 II		1	
	Comp Index 4		3	
	Branch Index Low		H	End of row?
	Add Index 3 II		J	
	Add Index 4 II		3	
	VFL Load Entry II		I-J+3	OFFSET = 0
	" ADM "		21, 0, 19	Prepare values for next row
	" " "		23, 0, 19	
	" " "		25, 0, 19	
	" " "		27, 0, 19	
	Comp Index 4 II		I(I-1)+1	
	Branch Index Low		H	
END				