NOTAT

TIL : DIVISION LEADERS
FRA : BL, TEP
DATO : 14/6-71
SAK : Time-sharing System

We suggest that there be made a Time-sharing system for the NORD-1 computer (NORD-1-TSS).

The purpose of a TSS would be to use the machine(s) at the data center in a more efficient way in the testing and debugging phases of programs by letting a number of people use a machine simultaneously. We believe that a TSS could also be of value for other NORD-1 users.

The TSS would be implemented in two phases. The first would be a simple TSS, which could be ready in a short time. In the next phase, the first version of TSS would be expanded to a more powerful system.

Implementation

The TSS would consist of 3 parts:

1) The time-sharing part

2) A file system to manipulate files

3) A monitor program (lets the user operate the computer from the teletype).
The monitor would execute commands for file manipulation, execution control, compiling, debugging aids, and utilities.

The time-sharing part could be taken from the multi-programming system developed at SI. This would require hardware paging on the NORD-1. The time-sharing part of the SI-system is working already, and could be used almost immediately, perhaps with some small improvements.

A simple time-sharing system where only one user is in core at a time is another alternative. The advantage of such a system is that it is simple and requires no new hardware.

A time estimate for either of these systems is 1 1/2 months. This assumes that TEP and BL would be working on it full time and that the hardware would be perfect.

The file system could be made from specifications held by BL. He has participated in the implementation of such a file system before.

It is possible to use the NORD-OPS file system. However, it is believed that it would take longer to integrate that file system into the time-sharing system than to implement BL's file system.

Below follows a description of a possible set of executive commands.

LOAD	loads a binary program from paper tape or a file and starts execution.
LOGOUT	releases the process associated with that teletype
(RUBOUT)	connect the TTY to the time-sharing system (i.e., log on)
RESET	clear the virtual machine
PLACE	like load except that the program is not started
GOTO	starts execution at the specified address
(RUBOUT)	pressing rubout while a program is running stops the program and returns to command mode
COPY	copy one file to another
LISTFILES	lists the files for a user
DELETEFILE	delete a specified file
EXAMINEFILE	examine a specified file
DUMP	dumps the current core on a file
RECOVER	recover a dump file

If we used the Si paging system and wanted 3 users, every user would have 4K (say) on a 16K machine with a 4K system. Recent experiments at SI have indicated that a user should have 4K (16 pages) in order to run the assembler efficiently.

In the simple swapping system, if each user gets 12K of core, the amount of time to swap one user out and another in is about 1/2 second. If one allows 1/2 second compute time for each user every 4 seconds then we can run 4 users. In other words it would take one second for each time a user is run.

four weeks. The two models can also be leased. TEKTRONIX, INC., Beaverton, Ore.

FOR DATA CIRCLE 332 ON READER CARD

High-speed Discs

Some members in this series of head-per-track disc units have 2.1-msec access times, easily qualifying them for applications requiring faster-than-average response times. An interface for Data General users is available now, with a DEC PDP-11 model in development. Capacities for the units range from 1-16 megabits. Reading is by non-contact recording heads mounted at three points around the disc for 2.1-msec access, at opposite sides for 4.2-msec, and at one point for 8.4-msec access, based on 3600-rpm rotation. The transfer rate is 4 megabits/second per track. Two representative models in the line-up are a 256K 16-bit word model with 4.2-msec access, priced at $5K, and a 512K 16-bit unit with 2.1-msec performance priced at $10K. Delivery is approximately 10 weeks. ALPHA DATA, INC., Canoga Park, Calif.

FOR DATA CIRCLE 334 ON READER CARD

Serial Printer

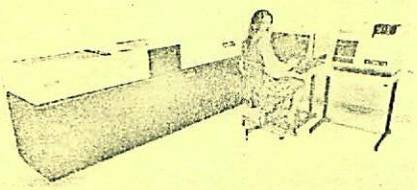
Centronics has added another serial printer to its product lineup. This unit, the model 500, prints a full 132 columns at 100 cps, or 40 full lines per minute, based on a 64-character ASCII set. All logic and control functions for the 500 are contained on a single LSI circuit board which might make for impressive reliability (or at least easy maintenance). Characters generated are 5 x 7 dot-matrix images.

Standard features include parallel data input at up to 75,000 cps, elongated boldface characters, paper runaway inhibit (10 second time-out), automatic line feed on carriage return, and printing of up to five-part tractor-fed paper. Options include an RS232 interface (100-9600 baud), vertical format control, an expanded character set, foreign character sets, and interfaces for a number of popular minicomputers. The 500 is priced at $2,600 and is available in 2-3 months depending on quantity. CENTRONICS DATA COMPUTER CORP., Hudson, N.H.

FOR DATA CIRCLE 335 ON READER CARD

Commo-oriented System

Another version of the highly-successful System 10 computer has been announced with enhanced communications capability that should be considered by larger corporations for use in a distributed computing network, or even by small businesses for their sole computer—the communications features provide a way to work with other computers in times of peak load. The model 101 is expandable from 10-30K of 3.3-usec core, and can have up to five I/O controllers for use with a variety of peripherals. A synchronous communications adapter permits data transmission to remote computers at 40.8 kilobaud rates, while operator-oriented ASCII-code terminals can talk to the 101 conversationally at rates up to 9600 baud. The same applications



program library offered with the larger System 10 works on this one, and up to five jobs can be executed simultaneously.

A card-oriented remote job entry terminal system including a 10K processor, card reader, crt, and line printer is priced at $40,360. A disc-oriented remote terminal processor system with 20K, a disc drive, crt, and line printer is priced at $61,285. Systems may also be rented. SINGER BUSINESS MACHINES, San Leandro, Calif.

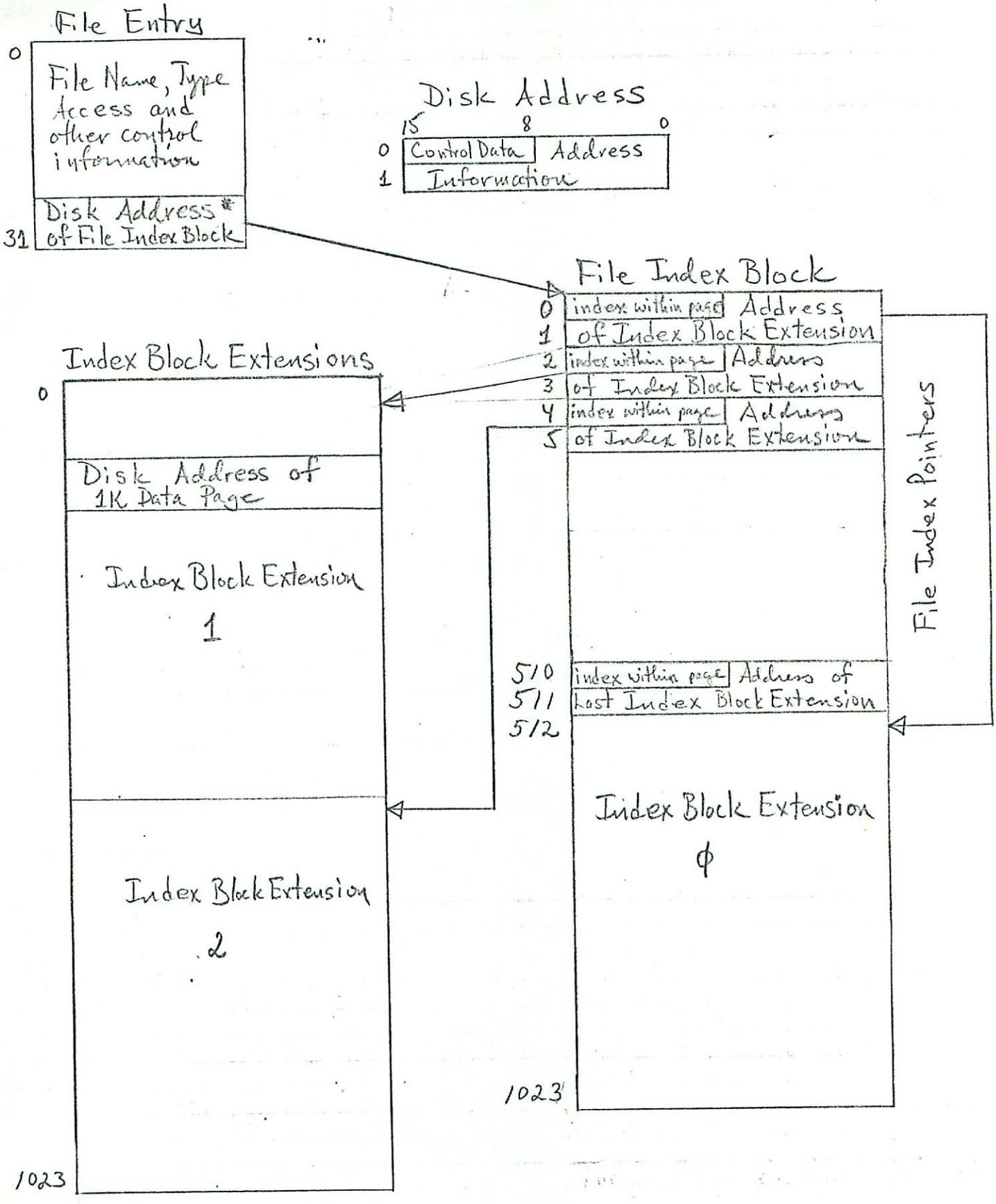
FOR DATA CIRCLE 336 ON READER CARD

S/3 Memory

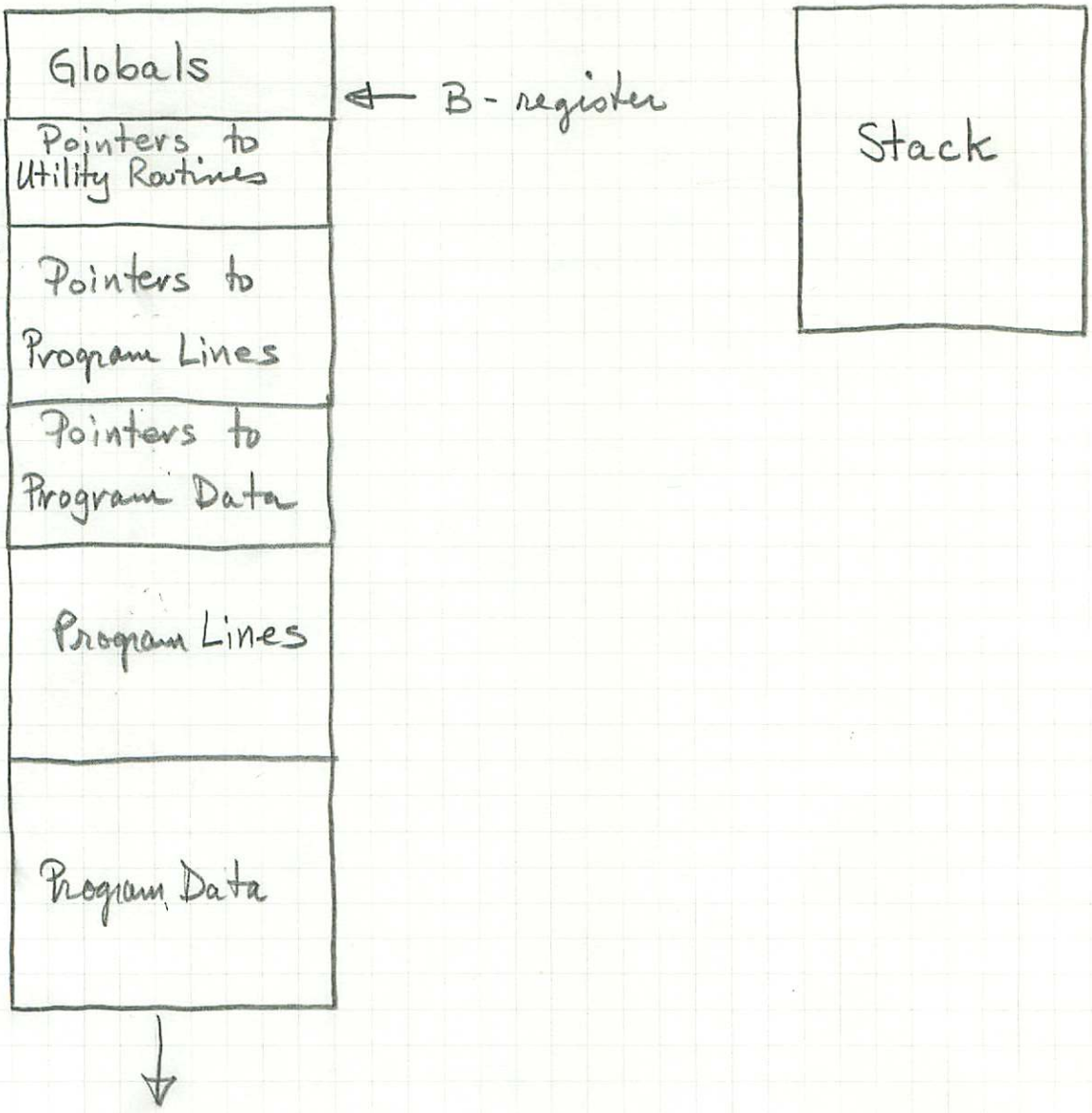
A firm that has been building products such as disc cartridges for the IBM System/3 has entered the main memory add-on market for that machine with a new twist: the memories use MOSFETS to augment the core on the S/3 models 6 and 10 just like that IBM recently introduced on the System/3 model 15. Several early installations of the product have shown no signs of a "technology conflict" according to the vendor, which also states that the memories are totally transparent to the user in all operations.

The memories are available in 8K increments for expansion of 8K systems up to a maximum of 64K. The first 8K chunk sells for $5,760 and rents for $169/month (including maintenance) on a two-year lease. For users who need the whole 56K shot, the figures would be $47,736 and $1,012/month, respectively. The memories will be marketed internationally, and domestic servicing for the products has been arranged with Sorbus. A switch is located on the memory that permits IBM service personnel to run diagnostics on the IBM portion of

Structure of a File (Updated 31/8-73)



* For absolute files (allocated contiguously) the pointer to the File Index Block will instead consist of the disk address of the file itself along with the size of the file.


Globals
B - register
Stack
Pointers to Utility Routines
Pointers to Program Lines
Pointers to Program Data
Program Lines
Program Data

New Instructions

PUSHF	Push TAD onto stack
PUSH dr	Push register onto stack
POPF	Pop TAD from stack
POP dr	Pop register from stack
READF	Read TAD from program variable specified by XREG.
READ dr	Read dr from program variable specified by XREG.
WRITF	Write TAD into program variable specified by XREG.
WRIT dr	Write dr into program variable specified by XREG.
GOTO	Goto program line specified by XREG.
GOSUB	Call subroutine specified by XREG.
ENTER	Enter recursive subroutine and allocate the amount of space specified by the next word.

LOADF	i	Load TAD from local variable i.
LOADT	i	Load T from local variable i.
LOADA	i	Load A from local variable i.
LOADD	i	Load D from local variable i.
LOADX	i	Load X from local variable i.
STORF	i	Store TAD into local variable i.
STORT	i	Store T into local variable i.
STORA	i	Store A into local variable i.
STORD	i	Store D into local variable i.
STORX	i	Store X into local variable i
INC	i	Increment local variable i.
DEC	i	Decrement local variable i.
RET		Return from recursive subroutine

8/1-73 Bo Jonsdal

Proposal for String Instructions

In the following, XREG will contain a string pointer in which bits 15 to 1 specify a word displacement and bit 0 specifies which half of the word (0: left, 1: right).

LDB (XREG, TREG)
XREG = string pointer
TREG = beginning of string

Load Byte

```
AREG ← contents(TREG + XREG SHR 1);
CNT ← (8 IF BIT0(XREG)=1 ELSE 0);
AREG ← AREG SHA CNT;
AREG ← AREG SHA ZIN SHR 8;
PREG ← PREG+1;
RETURN;
```

STB (XREG, TREG, AREG)
XREG = string pointer
TREG = beginning of string
AREG = byte

Store Byte

```
temp1 ← AREG;
temp1 ← temp1 SHA 8;
IF BITΦ (XREG) = 1 DO;
temp1 ← temp1 SHA ZAN SHR 8;
temp2 ← 177400;
ELSE DO;
temp2 ← 377;
ENDIF;
temp3 ← contents (TREG + XREG SHR 1);
temp3 ← temp3 AND temp2;
contents (TREG + XREG SHR 1) ← temp1 OR temp3;
PREG ← PREG + 1;
RETURN;
```

8/1-75 BS Revised

Proposal for Register Block Move Instruction

READ (XREG, AREG)
XREG = core location
AREG = interrupt level

Register Block Read

Moves to the 9 word register block for the interrupt level specified by AREG to the 9 word core block starting at XREG.

WRITE (XREG, AREG)
XREG = core location
AREG = interrupt level

Register Block Write

Moves the 9 word core block starting at XREG to the 9 word register block for the interrupt level specified by AREG. If AREG specifies the current interrupt level, then PREG will not be changed.

Proposal for MOVE Instruction

MOVE (TREG, XREG, AREG)
TREG = destination address
XREG = source address
AREG = count

Move Word

```
        IF AREG <= 0 DO;
L1:     PREG ← PREG + 2; RETURN;
        ENDIF;
        contents (TREG) ← contents (XREG);
        TREG ← TREG + 1;
        XREG ← XREG + 1;
        AREG ← AREG - 1;
        GOTO L1 IF AREG = 0;
        PREG ← PREG + 1;
        RETURN;
```

Drum = 210-217

Error Messages

```
MS1,    'NO MORE TRACKS AVAILABLE$\␣'
MS2,    'TOO MANY USERS$\␣'
MS3,    'USER ALREADY EXISTS$\␣'
MS4,    'NO SUCH USER$\␣'
MS5,    'NO MORE ROOM IN UIB$\␣'
MS6,    'FILE ALREADY EXISTS$\␣'
MS7,    'ILLEGAL OBJECT TYPE$\␣'
MS8,    'NO SUCH FILE$\␣'
MS9,    'OBJECT INDEX OUT OF RANGE$\␣'
MS10,   'NO MORE SPACE IN OBT$\␣'
MS11,   'NO MORE OFT ENTRIES$\␣'
MS12,   'NO ACCESS$\␣'
MS13,   'ALREADY OPEN FOR WRITE$\␣'
MS14,   'BAD FILE NUMBER$\␣'
MS15,   'NO SUCH TRACK$\␣'
MS16,   'BAD TRACK NUMBER$\␣'
MS17,   'TRACK ALREADY EXISTS$\␣'
MS18,   'NO SUCH PAGE$\␣'
MS19,   'FATAL ERROR$\␣'
MS20,   'TRANSFER ERROR$\␣'
MS21,   'BAD DEVICE TYPE$\␣'
MS22,   'DISK AREA ALREADY IN USE$\␣'
```

0 1 2 3 4 5 6

Calls to Basic File System

% Create User
% D = device ID
% X = pointer to user name
% Return A
% A = user number
% Failure Return A

✓ % Get User Number
% D = device ID
% X = pointer to user name
% Return A
% A = user number
% Failure Return A

✓ % Create File
% A = device ID
% X = pointer to file ID
(user no, Name, Ed, Type, Password, Object Type,
Log Blksz, Date.) = Full ID
% Return
% Failure Return A

✓ % Open File
- % A = device ID
% X = pointer to file ID
(user no, Name, Edition) = Short ID
% T = read-only flag
% Return A
% A = file number
% Failure Return A

✓ % Close File
% T = file number
% Return
% Failure Return A

✓ % Create Track
% T = file number
% A = track number
% X = number of tracks + type of allocation (upper 2 bits)
type of allocation: 0 ⇒ cylinder
1 ⇒ dynamic
3 ⇒ absolute

% Return
% Failure Return A

✓% Delete Track
% T = file number
% A = track no.
% X - number of tracks
% Return
% Failure Return A

✓% Delete File
% A = device ID
% X = pointer to file ID (short)
% Return
% Failure Return A

✓% Get Next Track
% T = file number
% A = track number
% Return A
% A = next track or -1 if no more
% Failure Return A

✓% Read Entry
% D = device ID
% X = pointer to file ID (short)
% T = pointer to array
% A = number of words (all if negative)
% Return
% Failure Return A

✓ % Read Object .
% D = device ID
% X = index of object
% T = pointer to array (first word must contain user number)
% A = number of words (all if negative)
% Return
% Failure Return A

✓ % Test if open routine
% A = device ID
% X = pointer to file ID (short)
% Return A if open
% A = file number
% Failure Return A

✓ % Change Name .
% X = pointer to old name (short)
% T = pointer to new name (medium)
(Name, Ed., Type, Password)
% A = device ID
% Return
% Failure Return A

✓ % Set File Access .
% A = device ID
% X = pointer to file ID (short)
% T = access word
% Return
% Failure Return A

✓ % Set UIB Access .
% A = device ID
% X = user number
% T = access word
% Return
% Failure Return A

✓ % Read OFT entry (like Read Entry)
% T = file number
% A = number of words
% X = pointer to array
% Return
% Failure Return A

✓ % Set Length
% T = file number
% AD = length
% Return
% Failure Return A

✓ % Read Length
% T = file number
% Return AD
% AD = length
% Failure Return A

✓ % Read MIB label
% T = device ID
% A = number of words
% X = pointer to array
% Return

✓ % Write MIB Label
% T = device ID
% A = number of words
% X = pointer to array
% Return

✓ % Check if pack in use
% T = device ID
% Return if not in use
% Failure Return if in use

✓ % Read Page

% T = file number

% D = ϕ ⇒ wait

% A = page no.

% X = core address

% Return

% Failure Return A

✓ % Write Page

% T = file number

% D = ϕ => wait

% A = page no.

% X = core address

% Return

% Failure Return A

✓ % Read Block Size

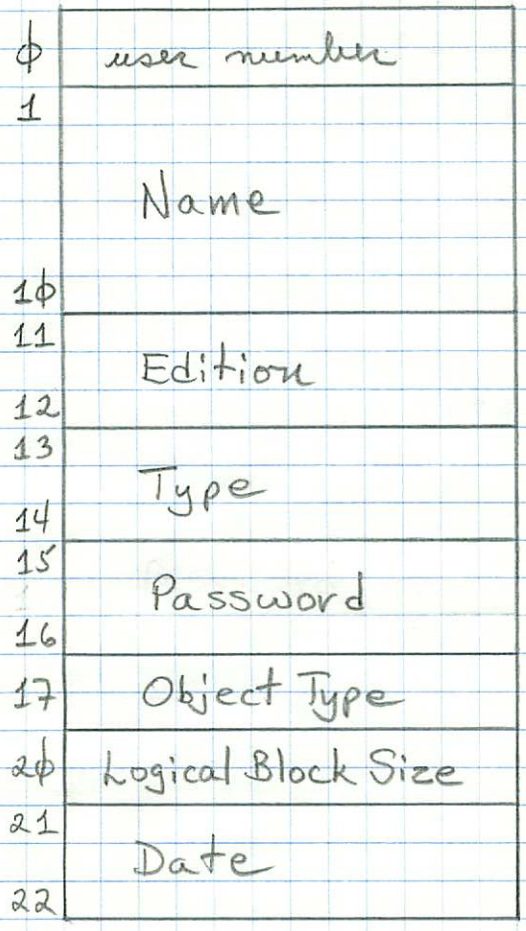
% T = file number

% Return A

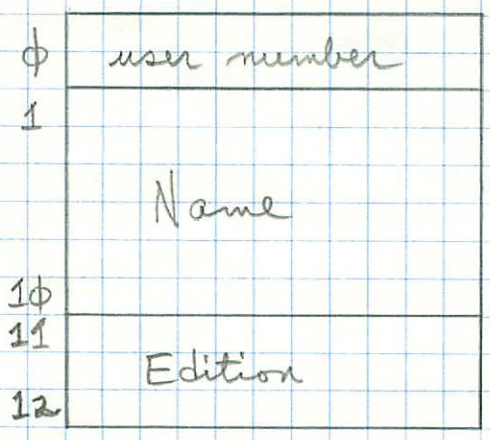
% A = block size

% Failure Return A

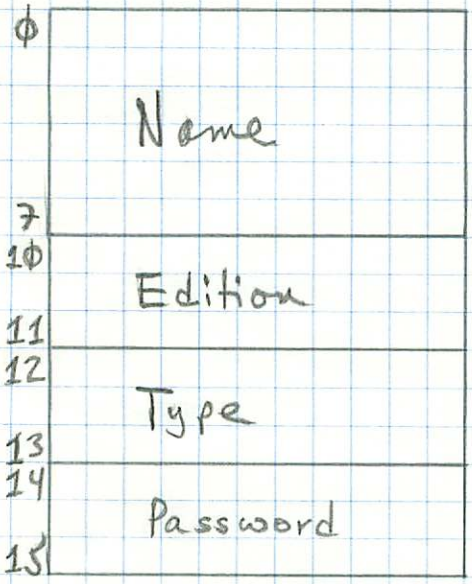
FULL ID


0
user number
1
Name
10
11
Edition
12
13
Type
14
15
Password
16
17
Object Type
20
Logical Block Size
21
Date
22

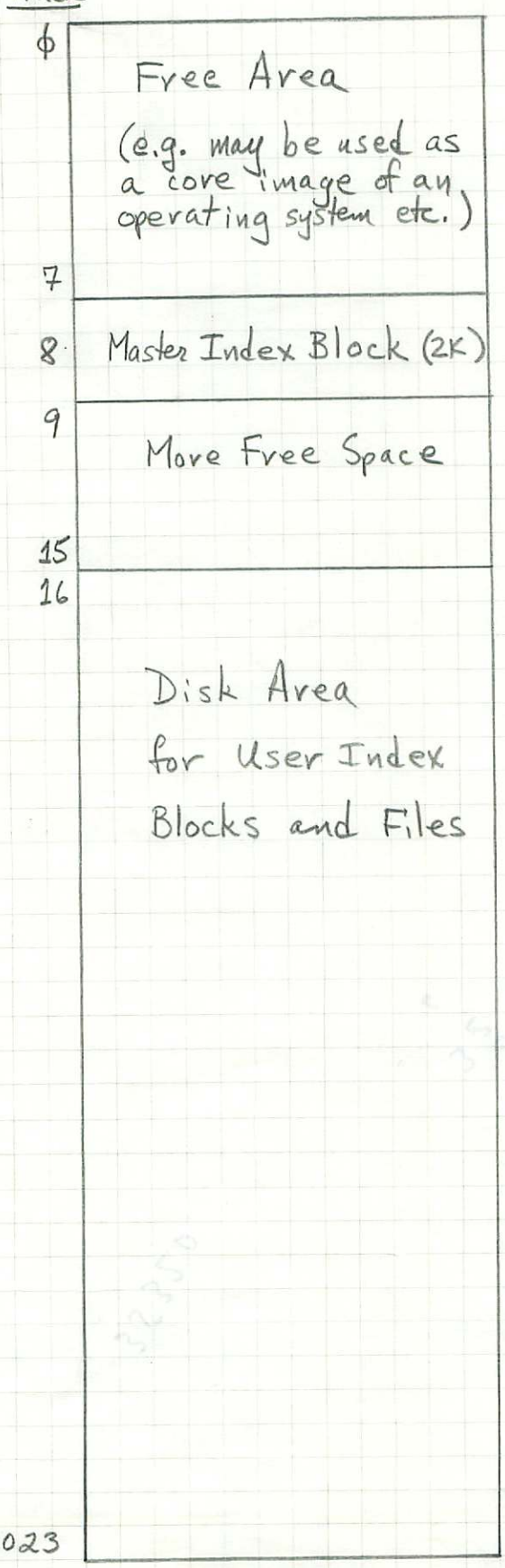
SHORT ID



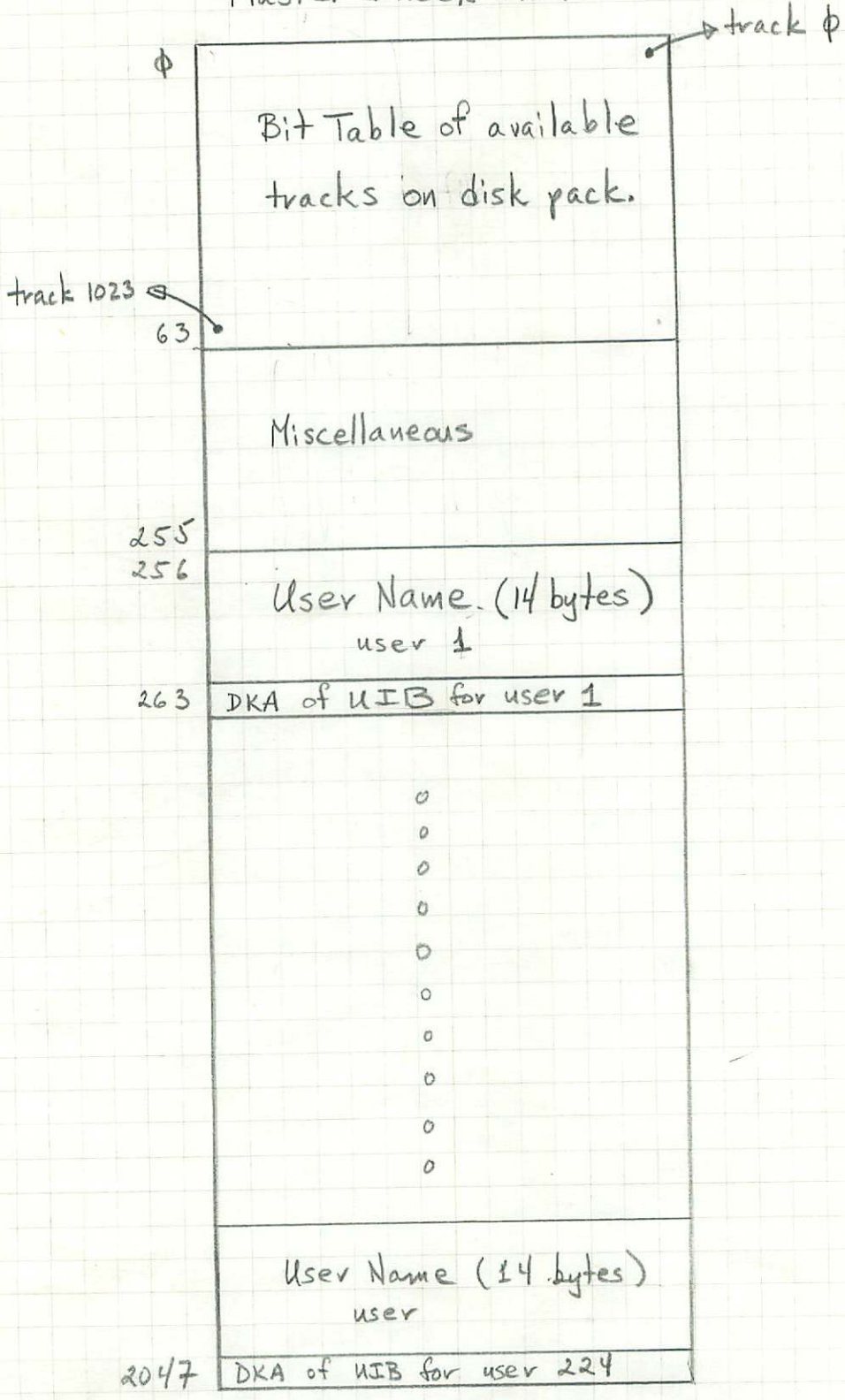
MEDIUM ID



Disk Layout


track
0
Free Area
(e.g. may be used as a core image of an operating system etc.)
7
8
Master Index Block (2K)
9
More Free Space
15
16
Disk Area for User Index Blocks and Files
1023

Master Index Block



User Index Block

0	Locked if nonzero			Lock on UIB
1		Owner O W R	Friend O W R	Public O W R	Access Word
2	Number of tracks available			
3	Friend 1	Friend 2		Friend Table
4	Friend 3	Friend 4		
5	Friend 5	Friend 6		
6	Free Area			
255				
256	Object Entry 0			
287				
2047	Object Entry ~~0~~ 55			

37600

40

Object Entry

at	Dec				
0	0	L	W	Count	
1	1	Name: 16 bytes			
4					
5					
10	8				
11	9	Edition: 4 bytes			
12	10				
13	11	Type: 4 bytes			
14	12				
15	13	Password: 4 bytes			
16	14				
17	15	Object Type	Owner O W R	Friend O W R	Public O W R
20	16	Device Type	Device No.		
21	17	DKA of Index Block			
22	18	File Length (in bytes)			
23	19				
24	20	Logical Blocksize			
25	21	Number of times used			
26	22	Number of times accessed			
27	23				
30	24	Year	Month	Day	Hr.
31	25	Hr.	Minute	Seconds	
32	26	user no.	object index		
33-36	27-30	Miscellaneous			
37	31	Displacement of index block segment			
40	32	Index Block Segment			
77	63				

Status Word
L: lock bit
W: write
Count: number of times file is open

Type is not considered to be part of the filename. Nor is Password.

Access Word

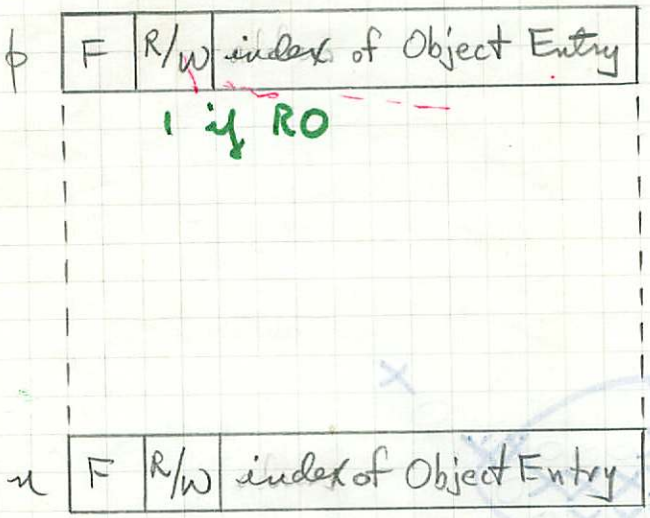
Medium

Date (Year as year - 1971)
This format due to J.E.Aa.

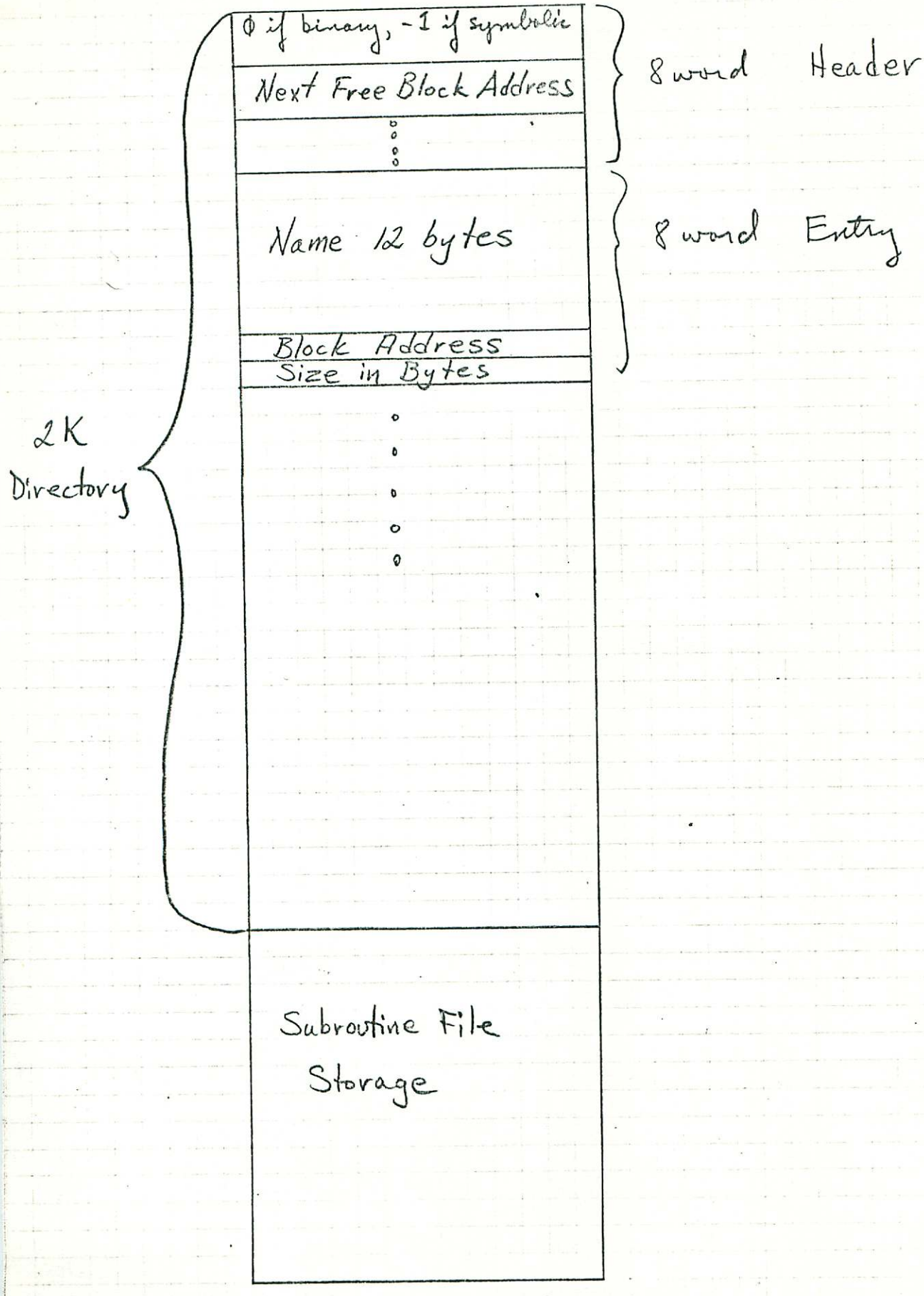
Only while in core

Word 33_8 => 0 normally, -1 if absolute file.

Open File Table



Subroutine Library File



Format of a Message

ϕ: type of message (ϕ for personal message, and 1 for broadcast)

1: number of characters in message (nch)
2: Sender
3 → (2+(nch+1)/2)): message string

Format of Message Area on Disk

ϕ: (next message)ϕ or -1 if none
1: addressee or -1 if empty
2 → 1+size(message): message

Note: Messages are never split up between two sectors; therefore the message string has a maximum length of 502 characters.

Also: Broadcast should set a flag preventing output by any user program. There should also be a lock on the MAIL file.

@MAIL → <all messages are output>
%HELP↵
%SEND-MESSAGE↵
DATE AND TIME: <arbitrary text>↵
TO: (USER) user name (may be abbreviated)↵
MESSAGE:↵
<arbitrary text which may be edited>
<terminated by &L> → DATE AND TIME: → ↵
%BROADCAST↵ (TYPE D or M:) if error
DIRECT OR TO MAILBOX? MESSAGE:↵
<arbitrary text which may be edited>
<terminated by &L>
%EMPTY-MAILBOX user-number or -1
%FINISH
@

Upon LOGIN, the following message will be output if the user has any mail:
<bells> "***YOU HAVE MAIL***" <bells>

TSS routines needed:

1.) Copy user's mail. (core address)
2.) Put message into user n's mailbox. (n, core address)
— 3.) Convert user name to user number. (STR: n)
— 4.) Send message to all active teletypes. (STR)
— 5.) Convert user number to user name (n: STR)

Bo Lewendal

Sak: Proposal for a Simple Timesharing System | 517 1971

This note consists of a proposal for a simple timesharing system to be used for the purpose of increasing throughput at the data center. The system would allow 3 or 4 users, each with a 12K address space, to use the machine concurrently. The 12K would range from 10000_8 to 37777_8. The system would lie in the lower 4K of core.

The swapping algorithm is very simple:

1.) Go to 3 if no user currently in core
2.) Transfer current user's context block and 12K of core onto mass storage.
3.) Transfer next user's context block and 12K from mass storage into core.
4.) Start user
5.) User running
   a.) If user's quantum runs out go to 2.
   b.) If user blocks for I/O or if he quits go to 2.

From the above one sees that swapping and execution cannot proceed simultaneously.

Sak: | 1/19

Following are the commands one may give to the timesharing system:

Q Quit

RESET	Clears the user's 12K address space.
LOAD	Loads a binary program from paper tape and starts execution of that program.
(RUBOUT)	Stops execution of the current program and gives control either to the command processor or a previously specified location in the current program.
PLACE	Same as LOAD except that the program is not started.
GOTO a	Transfers control to location a in the user's program.
DUMP n	Saves the current program on core load n.
CONTINUE n	Restarts the program which was previously DUMPed on core load n.
EXAMINE arg	Allows one to look at any register or core location. arg = P, A, D, T, X, L, B or S for STATUS or n for core location n_8.
SET n v	Set core location n_8 to v_8.

U Utility call

Following is a description of monitor routines callable from the user and the procedure for calling them:

A monitor call has the format:

arguments in the central registers

JPL I *+2

SUBCODE

MON

return, or failure return if the call has a failure return

return if the call has a failure return

SUBCODE is a number specifying the monitor call while MON is a constant specifying the normal entry point to the monitor.

Monitor Calls:

INBT T = file number
Reads a character from the specified file.
Failure return if end of file or error.

OUTBT A = character
T = file number
Writes a character onto the specified file.
Failure return if error.

EXIT — Return control to the timesharing command processor.

ECHO — A = echo mode

Set the echo mode to no echo if A=ϕ or to echo if A≠ϕ.

BREAK — T = break mode

A = break character

X = break table

Determine break strategy.

If T=ϕ then all characters are break characters else if T=1 then only control characters are break characters else if T=2 all characters are break characters except alphanumeric characters and space.

If T=3 then the character in A_{0-6} is defined to be a break character if $A_{15}=1$ or non-break character if $A_{15}=ϕ$.

If T=4 then X points to an 8 word bit table defining the break strategy for all characters.

OPEN — T = file number

Reserve the peripheral device given in T.

Failure return if device is already reserved.

CLOSE T = file number

Release the peripheral device given in T.

RDISK T = page number

X = core address

Reads page T into core at location given in X from the 64K area of the disk reserved for each user.

WDISK T = page number

X = core address.

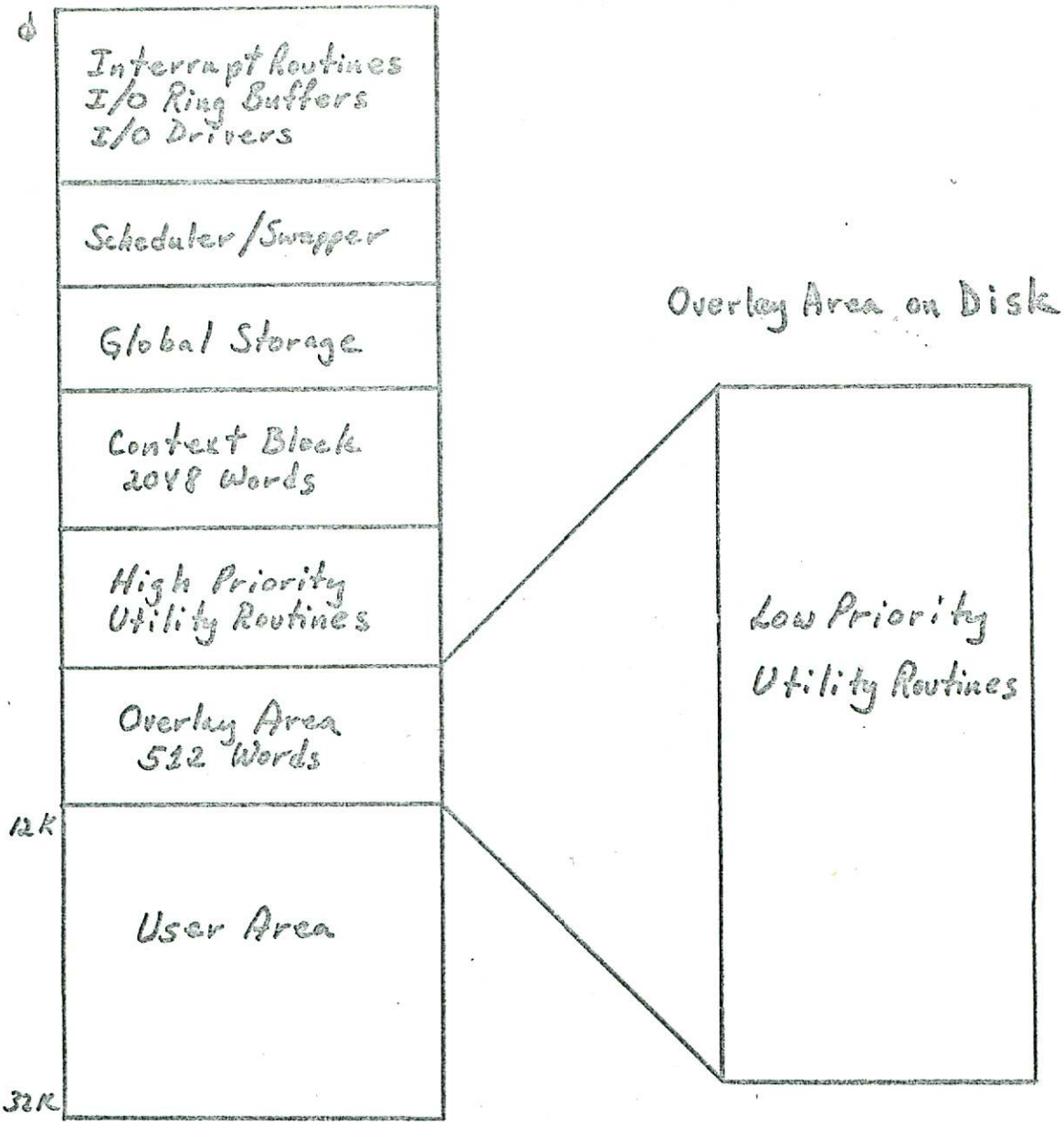
Writes page T from core.

Interrupt Levels Used by TSS 1.ϕ

15	Not used
14 -	Memory Protect Interrupt (May start level 3)
13 -	Clock Interrupt (Starts level 9)
12	Not used
11 -	Input Interrupt
1ϕ	Not used
9	Subsidiary Clock Interrupt (May start levels 6 and 5)
8	Not used
7 -	Output Interrupt
6	Teletype Scanner
5	Scheduler and Swapper
4	Not used
3	Decoding and execution of monitor calls (May start levels 5 and 1)
2	Utility Command Processor (May start levels 5 and 1)
1	User Program
ϕ	Traps WAIT instruction from level 1 (Starts level 2)

Note: Levels 1, 2 and 3 are considered to be a part of the user process and their level heads are therefore kept in the user's context block which is swapped along with the user's address space.

Core Layout of TSS



Syntax of Expressions

```
expression  = conjunction $(".OR" conjunction);
conjunction = negation $(".AND" negation);
negation    = ["NOT"] relation;
relation    = sum [(".EQ"/".GT"/".GE"/".LT"/".LE"/".NE")
              sum];
sum         = term $(("+"/"-"/".V"/".E") term);
term        = factor $(("*"/"/"/".LSH"/".RSH"/".LCY"/".RCY"/".A")
              factor);
factor      = ["-"/".N"] primary;
primary     = symbol / constant / "(" expression ")";
symbol      = macrocall / parameter;
constant    = numeral $ numeral [("B"/"D")[numeral]];
macrocall   =
```

Operators and Their Precedence

.OR	Boolean "or"
.AND	Boolean "and"
.NOT	Boolean "not"
.EQ, .GT, .GE, .LT, .LE, .NE	The Relations with values ϕ or -1
+, -, .V, .E	Add, subtract, logical "or", logical "exclusive or"
*, /, .LSH, .RSH, .LCY, .RCY, .A	Multiply, divide, left shift, right shift, left cycle, right cycle, logical "and"
-, .N (unary)	Unary negation, logical "not"

Sak: Description of Calls to file system | 15/6 1971

The syntax of a filename is as follows:

filename = ["(" user ")"] fname [(":" type [":" device] / "::" device)]

user = number / <name of a user>

fname = 1$8 (letter / digit / "-")

type = 1$4 (letter / digit)

device = 1$4 (letter / digit)

user is either a user number or a name which can be looked up in a table for its corresponding user number.

type would normally be one of the following, but could be anything: "SYM", "BIN", "PROG", "DATA", "DUMP".

device could be "DK1", "DK2", ... "DK4", "DR1", ... etc.

There will be special filenames associated with peripheral I/O equipment:

"PPT"	paper tape reader
"PUNCH"	paper tape punch
"CARD"	card reader
"TTYn"	teletype n
"PRINTER"	line printer

etc.

File System Calls

Open Write

ARG1: Pointer to filename (from teletype if -1)
If there are double quotes (") around the fname-type-device portion of the filename then the file is considered to be a new file.

Return fn (fn = file number)

Failure return if no access to file or if file does not exist

Open Read

ARG1: Pointer to filename (from teletype if -1)

Return fn

Failure return if no access to file or if file does not exist

Open Random

ARG1: Pointer to filename (from teletype if -1)

Return fn

Failure return if no access to file or if file does not exist

Close File

ARG1: fn

Failure return if no such fn

Write Byte

ARG1: fn

ARG2: byte

Failure return if no such fn

Holes in the file are not skipped

Write Word

ARG1: fn

ARG2: word

Failure return if no such fn

Holes in the file are not skipped

Equivalent to two Write Byte's

Read Byte

ARG1: fn

Return byte

Failure return if no such fn or if end of file

Holes are skipped in the file

Read Word

ARG1: fn

Return word

Failure return if no such fn or if end of file

Holes are skipped in the file

Set Pointer

ARG1: fn

ARG2: byte number (zero indexed)

Failure return if no such fn or if file is random

Read Pointer

ARG1: fn

Return byte number (zero indexed)

Failure return if no such fn or if file is random

Read Page

ARG1: fn

ARG2: file page number (zero indexed)

ARG3: core address

Failure return if no such file or if file is not random
or if no such file page

Write Page

ARG1: fn

ARG2: file page number

ARG3: core address

Failure return if no such file or if file is not random

Next Page

ARG1: fn

ARG2: file page number (may be -1)

Return the page number of the next file page after the given page. If no more pages, then -1 is returned.

Failure return if no such fn or if page number is out of range

Read Line

ARG1: fn

ARG2: terminating character or -1 for CRLF

ARG3: Pointer to where the line should be returned

Failure return if no such fn

Write Line

ARG1: fn

ARG2: Pointer to line

Failure return if no such fn

Read Length

ARG1: fn

Return 32 bit byte length of the file

Failure return if no such fn

Sak: Calls on Basic File System | 10/6 1971

1.) Create object

ARG1: pointer to name of object

~~ARG2: number of index blocks~~

Failure return if object already exists
no more room
no access
too many index blocks

2.) Delete object

ARG1: pointer to name of object

Failure return if object does not exist
no access

3.) Open object

ARG1: pointer to name of object

ARG2: read-only flag

Failure return if object does not exist
no access

Returns the file number

4.) Close file

ARG1: file number

Failure return if no such object is open

5.) Create page

ARG1: file number

ARG2: page number

Failure return if no such object is open
page already exists
page number out of range

6.) Delete page

ARG1: file number

ARG2: page number

Failure return if — no such object is open
no such page

7.) Get next page

ARG1: file number

ARG2: page number

Return next page number or -1 if no more

Failure return if — page number out of range

8.) Read page

ARG1: file number

ARG2: page number in file

ARG3: core address

Failure return if — no such page
no such object open

9.) Write page

ARG1: file number

ARG2: page number in file

ARG3: core address

Failure return if — no such page
no such object open
file is read only

10.) Read entry

ARG1: pointer to name of object

ARG2: pointer to an array

ARG3: number of words to be read

ARG3 words are read from the object entry into the array pointed to by ARG2.

Failure return if no such object name

11.) Read Object

ARG1: index of object in object table

ARG2: as in Read entry

ARG3: as in Read entry

Failure return if object table index is out of range

12.) Change name

ARG1: Pointer to old name of object

ARG2: Pointer to new name

Failure return if no such object name, new name already exists

13.) Set ~~length~~ reader pointer, writer pointer

ARG1: file number

ARG2: number of words in last page

Failure return if no such object open

14.) Read ~~length~~ reader pointer, writer pointer

ARG1: file number

Returns the length of the file as a 48-bit integer

Failure return if no such object open

15.) Set file access

ARG1: pointer to name of file

ARG2: access word

Failure return if no such file
no access

16.) Set UIB access

ARG1: user number

ARG2: access word

Failure return if no access
no such user

17.) Define friend

Access word:

Owner			Friend			Public		
R	W	O	R	W	O	R	W	O

Each user index block has an access word which defines the default access to its files.

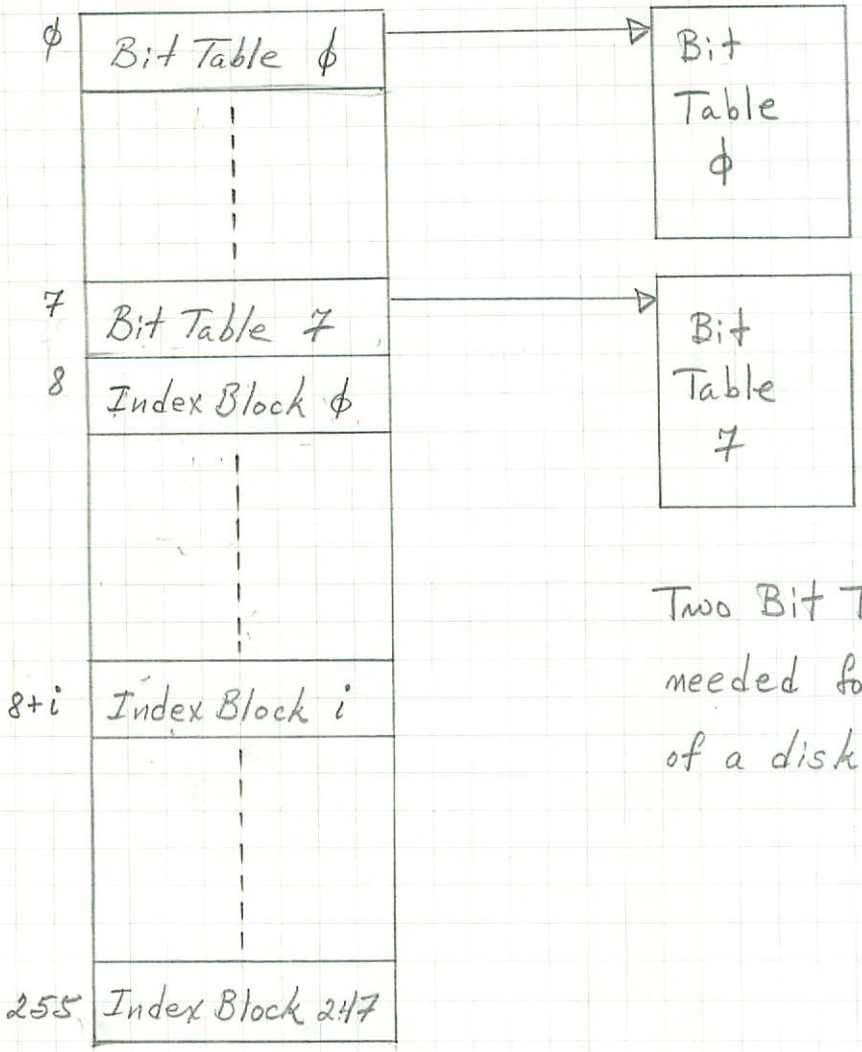
As each file is created, the default access is given to it. However, this access can be changed with the SET'FILE'ACCESS command.

The user index block access word may be changed with SET'UIB'ACCES.

The default value for the access word is

O(RWO), F(RW), P(R)

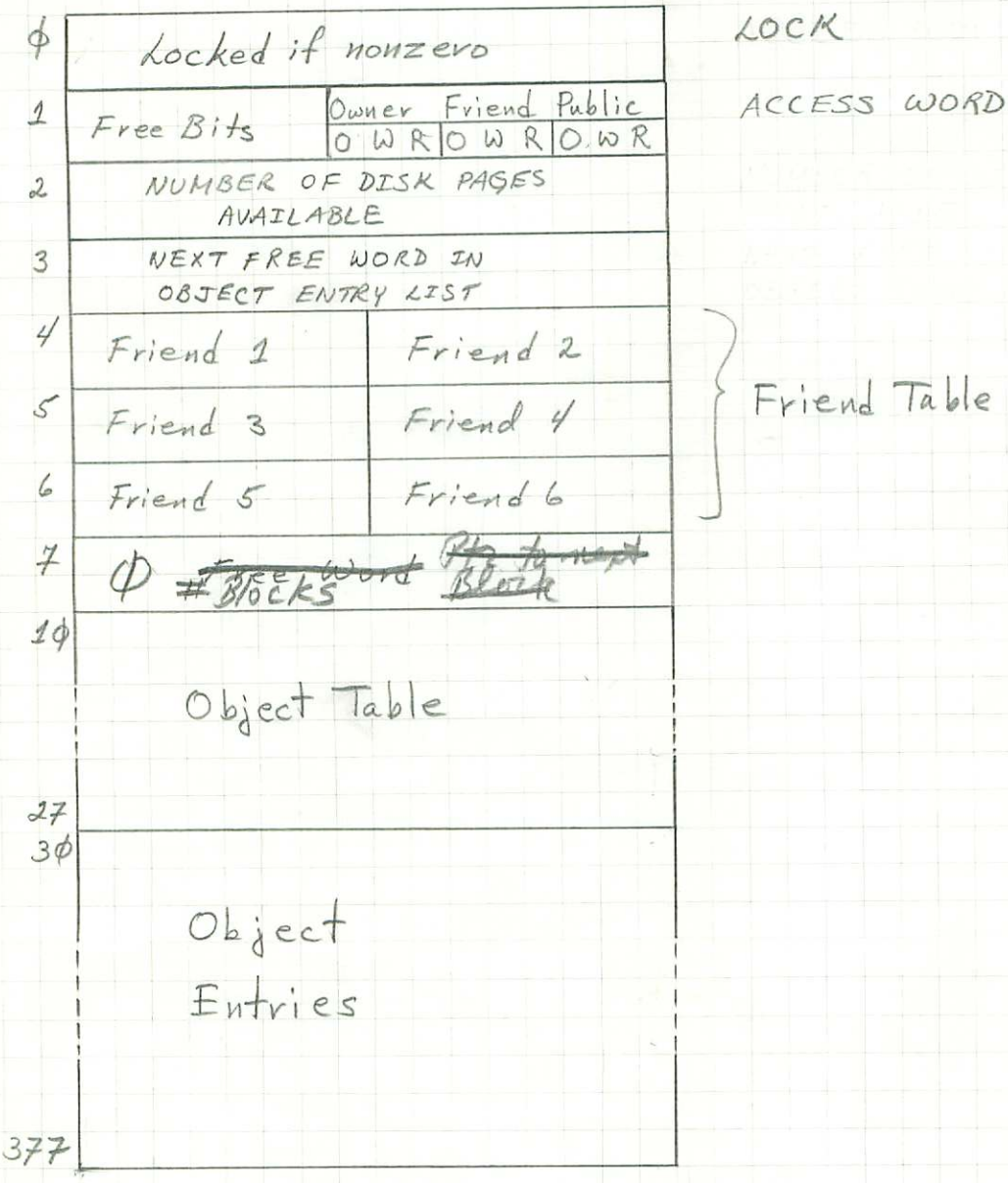
Sak: Master Index Block | 10/6 1971



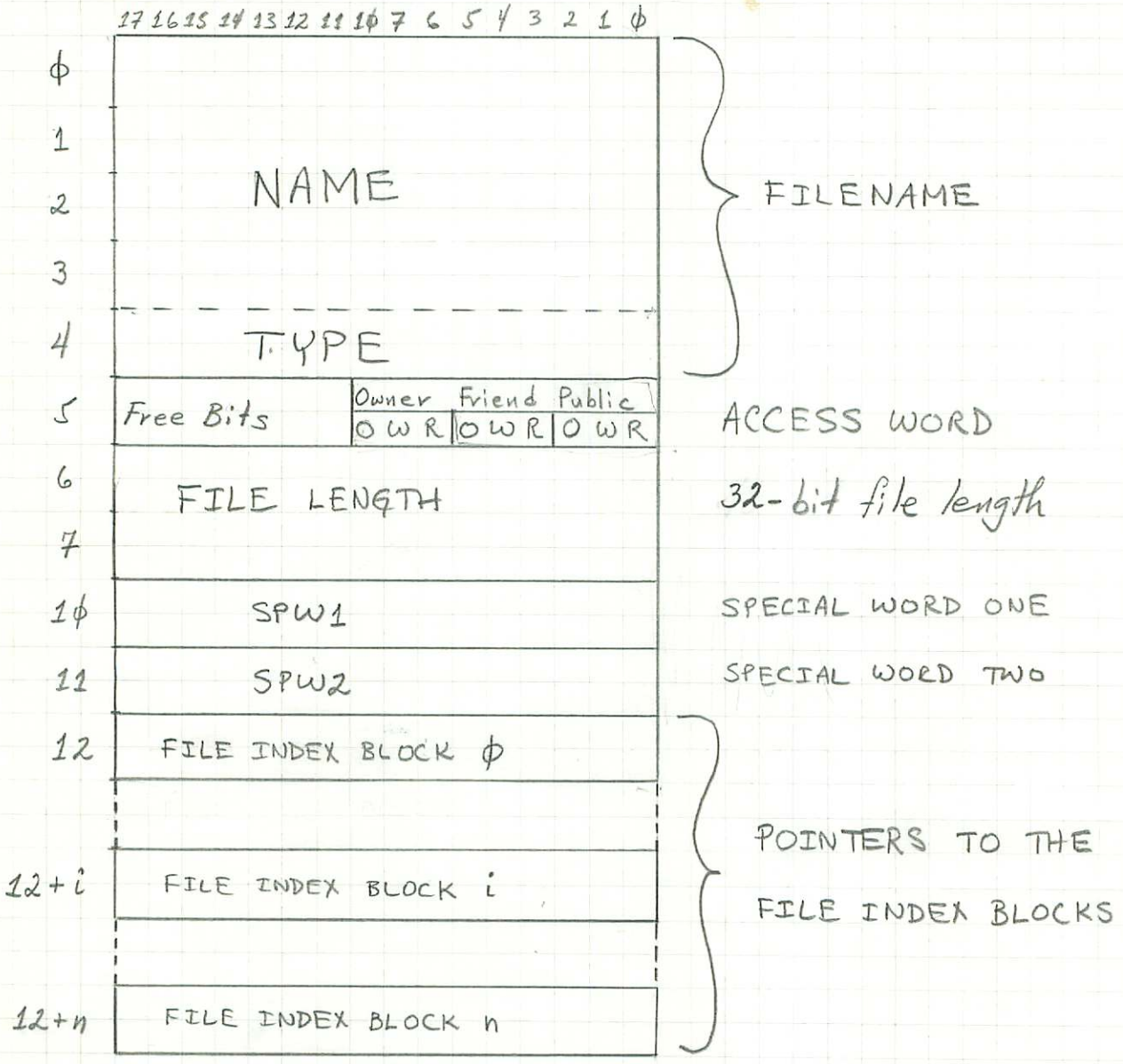
Two Bit Tables are needed for each spindle of a disk unit.

Note: There need only be two bit tables in each index block because it has been decided that there will be a Master Index Block for each spindle.

Sak: User Index Block | 10/6 1971



Sak: Format of Object Entry | 10 16 1971



Format of Object Table Entry

U	TYPE	SIZE	ADDRESS WITHIN USER INDEX BLOCK

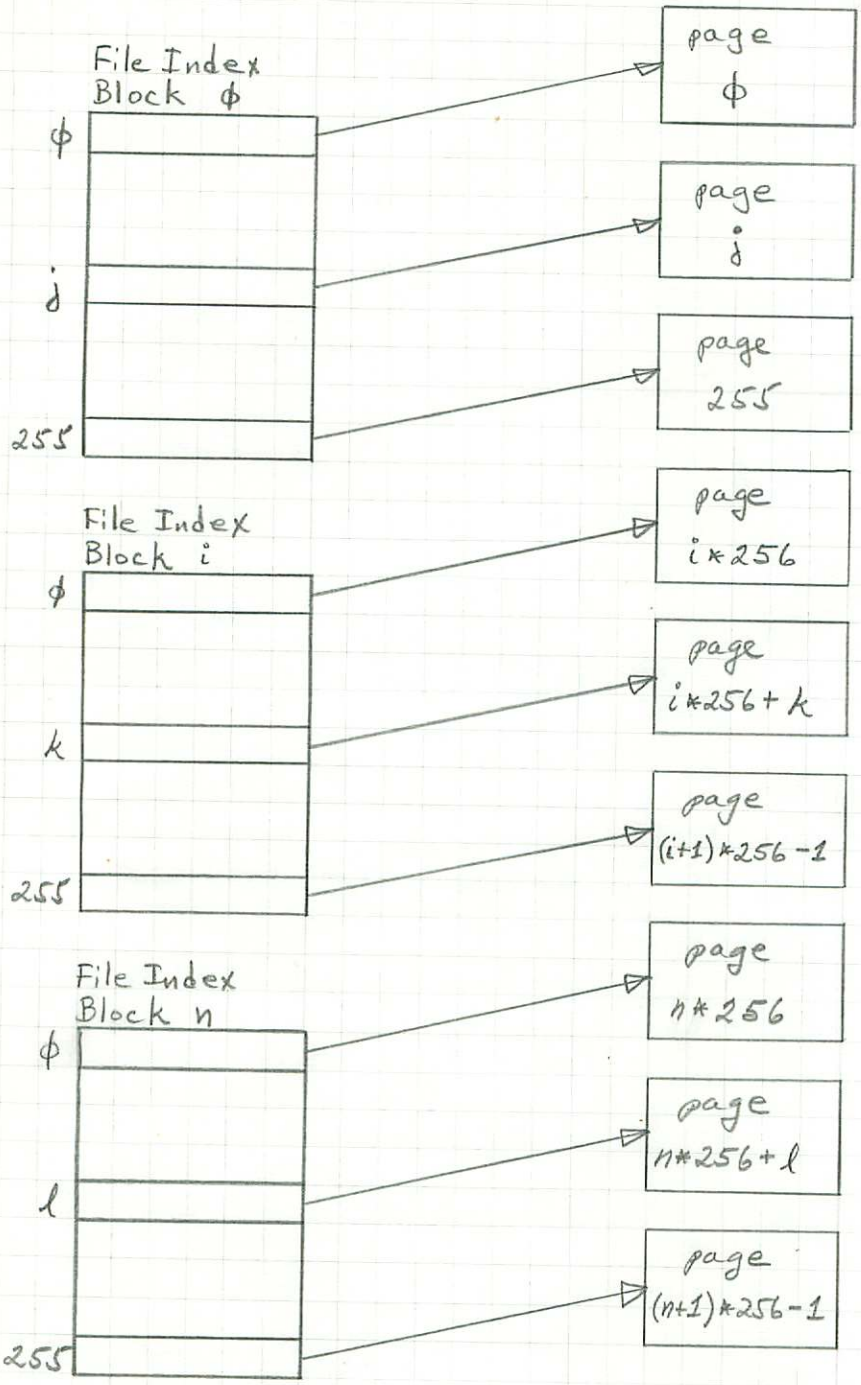
U: 0 if free else 1

TYPE: 0 = file
1 = process } not to be used initially
2 = data
3-7 = unused now

SIZE: The size of the object entry. In the above entry the size is 12+n. So we see that there is room for 6 file index blocks i.e. 1536 pages = 384 K words

Sak: File Index Blocks

10/6 1971



For each index block there are 256 pages = 64K words.

```
*MAIN
*Main instruction loop of CPU
*
MAIN:   M← PREG, N← 174000B, FETCH;
        IR← M← M AND N, RSH 11;
        GOTO M MRG IRTBL, N← 3400B, PREG← PREG+1;

*Instruction Table (must start on a 32 word boundary)

IRTBL:  M← IR, GOTO STZ;
        M← IR, GOTO STA;
        M← IR, GOTO STT;
        M← IR, GOTO STX;
        M← IR, GOTO STD;
          o
          o
          o
          o
```

```
* STZ     Store zero
STZ:      CALL DECOD;
          DATA ← Φ, STORE, GOTO MAIN;
* STA     Store AREG
STA:      CALL DECOD;
          DATA ← AREG, STORE, GOTO MAIN;
* STT     Store TREG
STT:      CALL DECOD;
          DATA ← TREG, STORE, GOTO MAIN;
* STX     Store XREG
STX:      CALL DECOD;
          DATA ← XREG, STORE, GOTO MAIN;
* STD     Store Double Word
STD:      CALL DECOD;
          DATA ← AREG, STORE;
          DATA ← DREG, M ← M+1; STORE, GOTO MAIN;
                 :
* MIN     Memory increment
MIN:      CALL DECOD;
          FETCH;
          DATA ← M+1, M ← EL, STORE, GOTO MAIN;
```

```
* LDA    Load AREG
LDA:    CALL DECOD;
        FETCH;
        AREG ← M, GOTO MAIN;

* LDD    Load Double Word
LDD:    CALL DECOD;
        FETCH;
        AREG ← M, M ← EL+1, FETCH;
        DREG ← M, GOTO MAIN;
*ADD    Add memory to AREG
ADD:    CALL DECOD;
        FETCH, Q ← AREG;
        AREG ← M+Q, GOTO MAIN;
*SUB    Sub memory from AREG
SUB:    CALL DECOD;
        FETCH, Q ← AREG;
        AREG ← NOT M + Q + 1, GOTO MAIN;
*ORA    Merge memory and AREG
ORA:    CALL DECOD;
        FETCH, N ← AREG;
        AREG ← M OR N, GOTO MAIN;
```

```
* JMP    Unconditional Branch
JMP:    CALL DECOD;
        PREG ← M, GOTO MAIN;

* JPL    Branch and save PREG
JPL:    CALL DECOD;
        Q ← PREG;
        PREG ← M;
        LREG ← Q, GOTO MAIN;
```

```
* DECOD
* Decode NORD-1 address
* M = (IR)
* N = 3400B
* Return
* EL = effective location
* M = (EL)
DECOD:  Q← M AND N RSH 8, N←7;
        GOTO Q AND N MRG DECBR, N←377;

* DECBR must be on an eight word boundary.
DECBR:  M←IR, Q←PREG, GOTO DEC0;
        M←IR, Q←BREG, GOTO DEC0;
        M←IR, Q←PREG, GOTO DEC1;
        M←IR, Q←BREG, GOTO DEC1;
        M←IR, Q←XREG, GOTO DEC0;
        M←IR, Q←BREG, GOTO DEC2;
        M←IR, Q←PREG, GOTO DEC3;
        M←IR, Q←BREG, GOTO DEC3;
```

* Address relative to P, X or B

```
DEC0:   M←M AND N LSH 24;
        M←M RSH 24;
        EL←M+Q, M←IR, RETURN;
```

* Indirect addressing or pre-indexing

```
DEC1:   M←M AND N LSH 24;
        M←M RSH 24;
        M←M+Q, FETCH;
        EL←M, M←IR, RETURN;
```

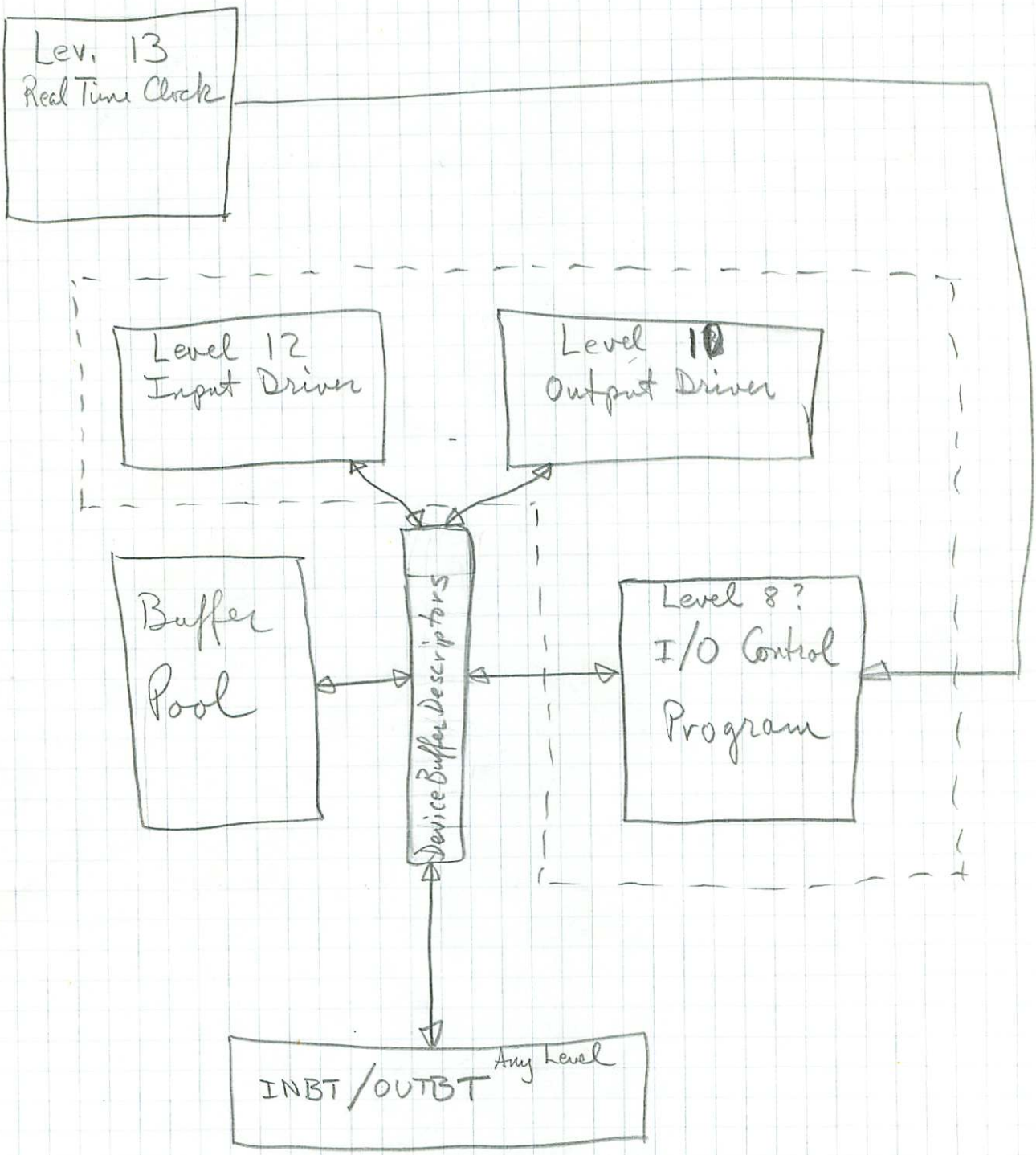
* Address relative to B and X

```
DEC2:   M←M AND N LSH 24;
        M←M RSH 24, N←XREG;
        M←M+Q;
        EL←M+N, M←IR, RETURN;
```

* Post-indexing and Pre- and Post indexing

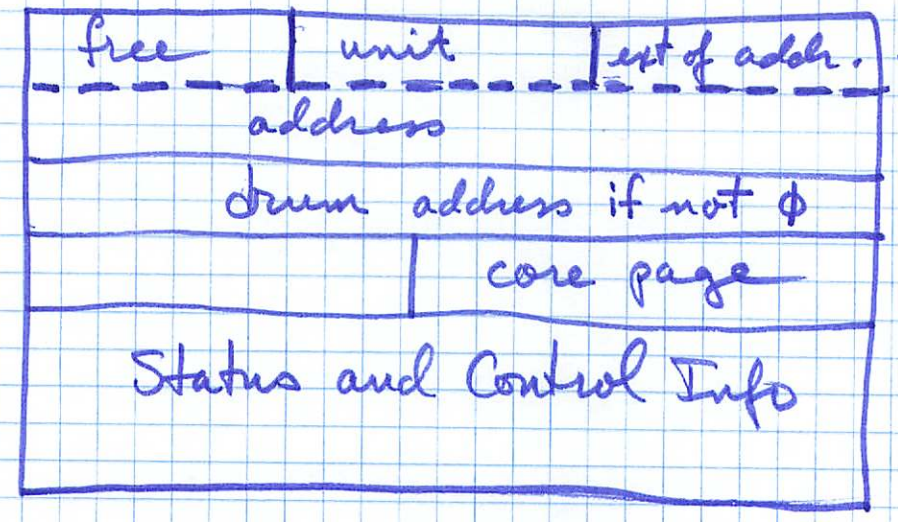
```
DEC3;   M←M AND N LSH 24;
        M←M RSH 24;
        M←M+Q, N←XREG, FETCH;
        EL←M+N, M←IR, RETURN;
```

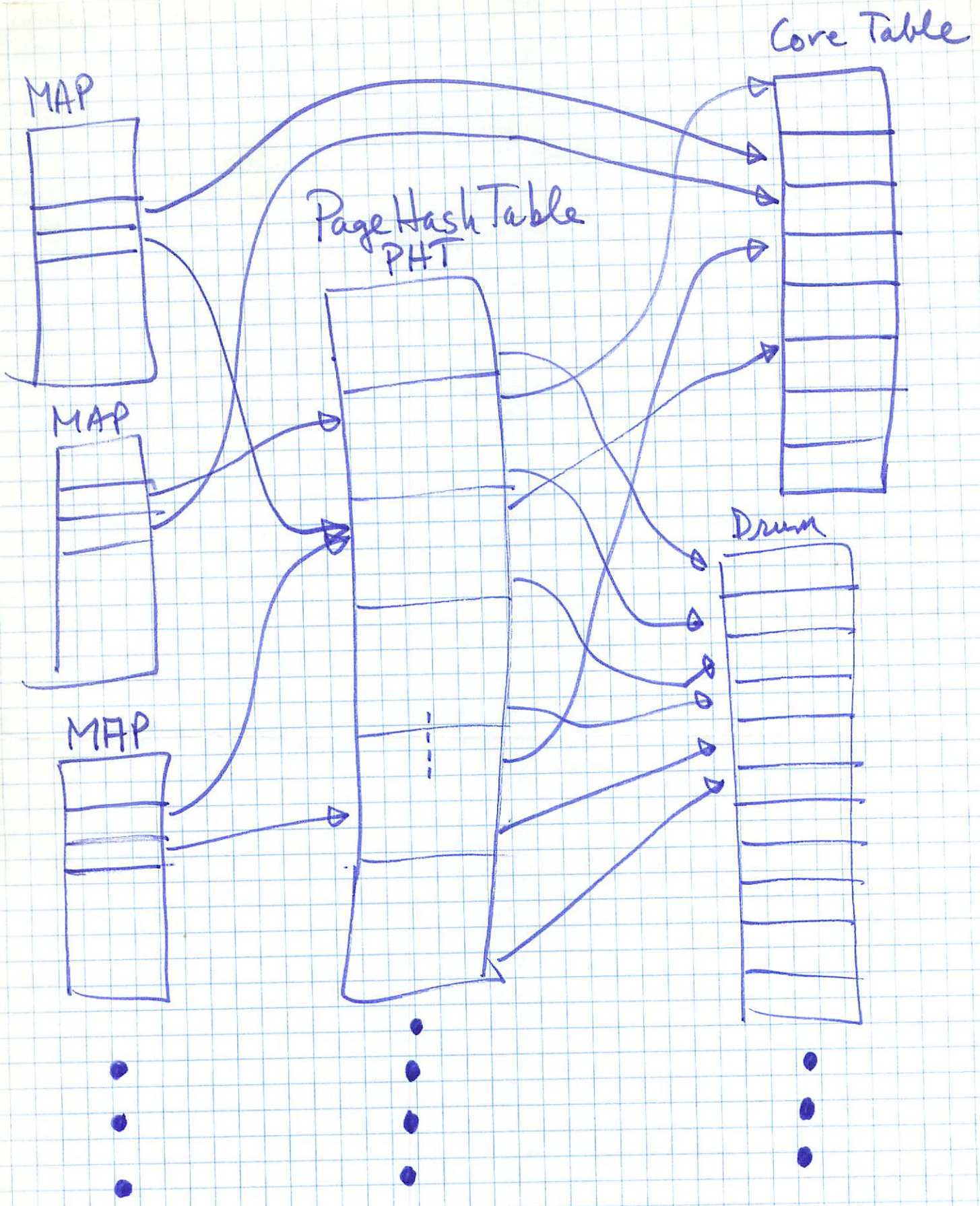
NORD-10 I/O System



The modules within the dotted box may be located in a separate CPU. In that case the real time clock may be omitted if the I/O CPU is to do only I/O.

PHT Entry


free
unit
ext of addr.
address
Disk Address
drum address if not ϕ
core page
Status and Control Info


Core Table
MAP
Page Hash Table
PHT
MAP
Drum
MAP

S_{min} = 10 ms. 7 10 1K pages

S_{max} = 70 ms. 70 55

W = 400 ∞

T = 25.5 ms. ~~16.7~~ 16.7/10 = 2

n = 10 - 50. 40

	E[s]	E[a]	E[t_s]	W_F	U	Q	
FCFS	30.5	43	51	n * 51	15.7	19.6	
$SCAN_{10}$	15.5	28	36	n * 24	22.2	27.8	
$SCAN_{20}$	12.9	25.5	33.5	n * 22.3	23.9	29.9	
$SCAN_{30}$	11.9	24.4	32.4	n * 21.6	24.7	30.9	57%
$SCAN_{40}$							
$SCAN_{50}$							
$SCAN_{60}$							
$SCAN_{70}$	10.8	23.3	31.3	n * 20.9	25.6	31.9	

17.5 FCFS

34.4 SCAN(40)

$SCAN_{40}$	8.5	21.25	29.8	n * 19.9	28.5	33.6
$SCAN_{40}$	11.1	19.5	21.5	n * 14.3	9.3	46.5

SCAN

$$E[s] = S_{min} + \frac{S_{max} - S_{min}}{n+1}$$

$$E[a] = E[s] + \frac{T}{2}$$

$$E[t_s] = E[a] + \cancel{\frac{T}{4}}\ \frac{T}{3}$$

$$W_F = \frac{2}{3} n\, E[t_s] \qquad U = \frac{t}{E[a]+t} = \frac{t}{E[t_s]}$$

$$Q = \cancel{\frac{1}{E[t_s]}}$$

FCFS

$$E[s] = \frac{W-1}{W}\left[S_{min} + \frac{1}{2} + \frac{S_{max} - S_{min}}{4} \cdot \left(1 + \frac{1}{3}\left(\frac{W}{W-1}\right)^2\right)\right]$$

$$E[a] = E[s] + \frac{T}{2}$$

$$E[t_s] = E[a] + \cancel{\frac{T}{4}}\ \frac{T}{3}$$

$$W_F = n\, E[t_s]$$

$$Q = \cancel{\frac{1}{E[t_s]}}$$