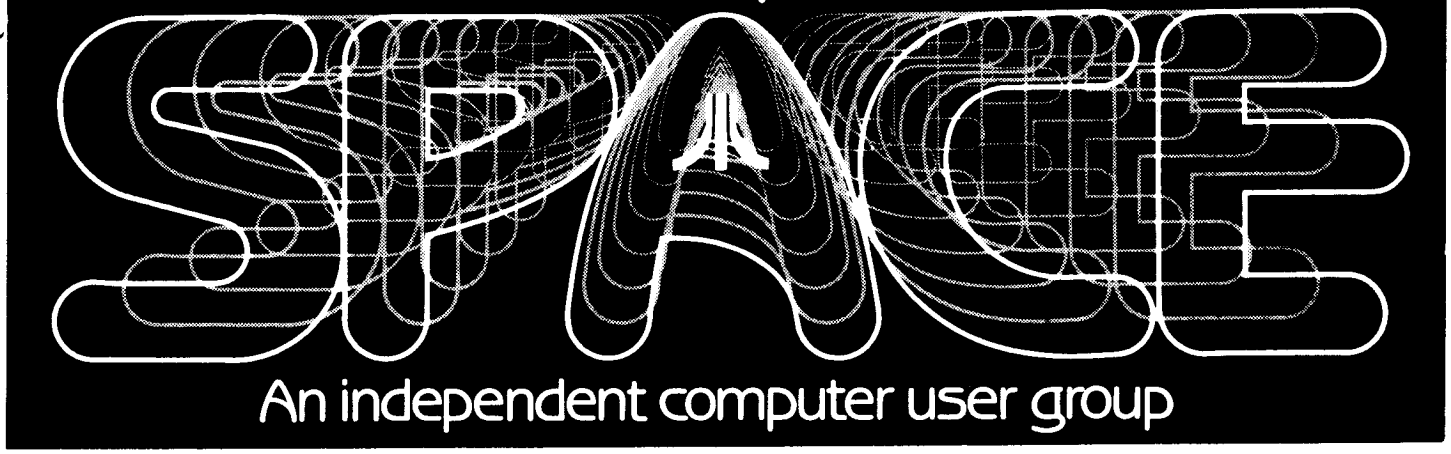
Saint Paul ATARI Computer Enthusiasts



An independent computer user group

NOVEMBER 1985

Next meeting: Friday, November 8, 7:30 PM

We will hold elections at this meeting.

Be sure to notice the new meeting place. Map on back cover.

December meeting: Friday, December 13

Bob Floyd . . .	President	487-2627	Jon Nelson . . .	Editor	484-9027
Bruce Haug . . .	Vice-Pres	774-6226	Max Feuer. . . .	Treasurer	483-3895
Frank Haug . . .	Disk Lib.	774-6226	Bruce Haug . . .	Cass Lib.	774-6226
Jim Schulz . . .	Paper/Ed. Lib.	537-5442	Steve Pauley . .	Secretary	560-2917

SPACE/TAIG Bulletin Board . . . 473-2897

Published by the St. Paul Atari Computer Enthusiasts (SPACE), an independent organization with no business affiliation with Atari Incorporated. Permission is granted to any similar organization with which SPACE exchanges newsletters to reprint material in this newsletter. We do, however, ask that credit be to the authors and to SPACE.

IN THIS ISSUE:

BYTES FROM THE PRES.
by Bob Floyd

Who was that masked man? - I wasn't sure whether the last meeting was "Shootout at the OK corral" or a Jack Tramiel testimonial. I think the last meeting turned out rather poorly. We didn't get to see the promised demos from ATARI rep Charlie Devine and he didn't answer any of our pre-selected questions. It was obvious that Charlie did not intend to demo anything, including the Casio keyboard he promised only 2 days before the meeting to bring (we lucked out and had one there anyway). Don't figure on our inviting him back. Also, the ambushers of the rep could have waited until after the meeting to spring their trap. (I'm not sure this would have worked anyway since Charlie beat a hasty retreat). I'm glad that meeting is over.

Elections this month - We will have elections this meeting. Look for the separate article by the nominating committee on who is running for what office.

New meeting room - Be sure to come to our new meeting room this month. We are nearby the old location, only 3 blocks away. The meetings are in the Falcon Heights Community Center at 2077 West Larpenteur (near Gortner). The police and fire station are in the same building and a Hewlett Packard office is next door. See the map elsewhere in the newsletter. Hopefully this a meeting room we can stick with for a while.

Meeting agenda - We will try not to spend too much time talking this month and spend more time on demos and swap meet. We won't spend much time on ST stuff since some decision regarding mini-groups or new group for ST is imminent. (Bruce Haug and myself are having discussions with others concerned and will figure something out soon. The club will remain loyal to the 400/800/XL/XE series).

Swap Meet - There will be a swap meet this month after the meeting. This is for buying and selling of used software and hardware. If you are selling software, you must have the original documentation and software - no pirating!

NICELIST - Remember the program lister we offered to the club a few months ago? Apparantly ATARI likes NICELIST enough that they now use it for program listings in the "ATARI Explorer". They called me up to thank me for the program and have promised to mention SPACE in the near future.

Bye-Bye - This is my last "Bytes", so I just wanted you all to know that I've enjoyed being "the pres" and plan to stay active in the club. I think that the club has done well in the last 16 months and I'm sure this trend will continue with the new officers we will be electing. Thanks to all who helped along the way!

ST's WHAT TO DO
BRUCE HAUG V.P.

All we hear is ST this, and ST that! Well it's about time all the ST owners get together and form a SPACE 520ST SIG, that is a part of SPACE. The SPACE officers are willing to help get a group started, but not run it. The group can decide for itself what they want to do, such as: meet before, or during the SPACE meeting, like the BEGINNERS GROUP, or on another night at some other place. The SPACE 520ST SIG would have input into the main SPACE meeting, have access to the newsletter to print ST NEWS, and have money to start a ST disk of the month. I'am sure all the details can be worked out with the newly elected officers of SPACE.

By doing the ST as a Special Interest Group (SIG) the members of SPACE can have the best of two computing worlds, The EIGHT Bit, and THE SIXTEEN bit, With both included in one membership fee, and one newsletter.

The rest of us can get back to the world of the EIGHT BIT machine (the older and newer Atari's) and enjoy the meeting without it being devoted to ST NEWS. This does not mean we will not hear about the ST, but the meeting will be more old Atari than the new ST.

Rumor has it that User Friendly has already started a 520 ST group, come to the meeting and find out whats going on. I hope there will be enough interested ST owners at the meeting to get something started, in fact they may have lot of things to bring to the SWAPMEET, so everyone should come. SEE YOU FRIDAY.

October Meeting Minutes
by Joanne Floyd

The last SPACE meeting was held Friday, October 11, with 86 people in attendance. In discussing the latest news from ATARI, Bob Floyd cited an Infoworld article which reported that the GEM operating system in the 520ST will be altered to avoid lawsuits with Apple. In the president's report, Bob discussed reasons for the move to the new meeting room at Falcon Heights Community Center. In the vice president's report, Bruce Haug reminded members that "C" language disks are available from the club for $4; he also recommended **Going From Basic to C** and **The C Programming Guide** as good resources to buy to learn the language. (Both books are available from the paper library.) Max Feuer, the treasurer, reported that the club has a bank balance of $960.

Dick Johnson, the president of TAIG, reported that a 130XE has been purchased for the BBS, and that the SYSOP is trying to structure the board to allow higher access to TAIG and SPACE members. In new business, the nominating committee announced nominations for the offices of president, secretary, and treasurer and Jim Schulz was nominated from the floor for the office of vice president. Bob Floyd mentioned that a newsletter editor would be needed

to replace Jon Nelson. (He noted that special thanks should be given to Jon for the great job he's done with the newsletter.)

At the end of the meeting, Charlie Devine, the local ATARI rep, spoke to the group. His talk was long on unconvincing accolades to Jack Tramiel and short on useful demonstrations of new products. He did mention (unofficially) that there will be no new 8-bit line from ATARI since they consider the future of computers to be in the 32-bit line. He predicted that the operating system upgrade for the the 520ST would be available within 60 days and that IBM compatability would be available for under $300 within 4 or 5 months. At the end of his talk, Charlie asked for complaints about ATARI but was not very helpful or informative in dealing with the complaints.

D.O.M. News

by: Frank Haug

Well this month's disk may not be the fullest but I'm glad that ALL of the programs WERE member donations. (I knew you guys could do it, you just had to roll up your sleeves and dig in.) Thanks, but you other people don't get off the hook so easily.

Speaking of donations... I have an idea that might work out nicely for us. I'm sure some of you own some sort of graphics program for drawing pictures. Now some of these pictures are pretty good. If enough people donate pictures to make it worth selling, a disk containing these files could be sold as a separate disk. On that same thought those of you who have Electronic Arts' Pinball Construction Set could donate the games you made for another separate disk (The Games themselves not files loaded by it for its use) This will all depend on your support. If you're interested get in touch with me at the meeting.

Besides the DOM we will be selling a separate disk for $4.00. This disk is a S.A.M. tutorial. You must have S.A.M. in order to use it. (For those of you who never heard of S.A.M., it stands for Software Animated Mouth. It is a speech synthesizer that doesn't require extra hardware.)

Now on with the programs on the November 1985 D.O.M.

1.EDIT.INS - Run this program for the instructions to EDIT.

2.EDIT - A member-written Screen editor.

3.SUMO.BAS - A text simulation of a Sumo-wrestling match.

4.DRAG.BAS - A text simulation of a drag race.

5.SCROLL - ENTER program and type RUN (Note your program can't use line 0 or lines from 32710 on) to allow you to scroll up and down the listing via - or =.

6.PINBALL.INS - Run this program for the instructions to pinball.

7.PINBALL.OBJ - Will NOT show up on menu, must be loaded without basic from DOS. RUN PINBALL.INS first.

NOTES FROM THE EDITOR

by Jonathan Nelson

I would like to thank all of you, especially the other officers, for all the support you have provided in the last sixteen months. You have provided each other with better, more interesting news. I hope you keep on supporting the editors-to-come as much as you did me; it will make his job easier and your newsletter more exciting. So long for now.

The Ed

NOMINATIONS COMMITTEE REPORT

By Sherm Erickson
and Glen Kirschenmann

The following people have either been contacted or turned their names into the Nominations Committee:

Pres. : Bruce Haug
V.P. : Jim Shulz
Secty.: Bob Floyd
Treas.: Bob Siede

Nominations for office will be open before the Elections begin, so if you did not turn your name into the Nominations Committee you still have a chance to run for office. See you at the NEW MEETING PLACE, SEE MAP IN THIS NEWSLETTER.

USING DISKS WITH BASIC (FOR THE CURIOUS ONLY)

By Chris Copeland
PART II

Last month we learned how to OPEN and CLOSE disk files and read and write to them. This month we'll look at two powerful disk commands- NOTE and point. They allow you to read and write to any place in the same file. If you have a file opened and say

```
NOTE #1,SECTOR,BYTE
```

in a program line, the number of the current sector and byte being read from or written to will be stored in the variables SECTOR and BYTE (You can use any variable name).

A sector is the basic unit of data on a disk. There are 720 on a DOS 2 disk. Each one holds 125 bytes of data. When you NOTE #1, SECTOR, BYTE you know in which sector your record is and on which byte of that sector it begins.

If you are using DOS 3, your disks have more than 720 sectors. They are grouped in "blocks".

When you know the sector and byte at which each record begins, you can then POINT to it if you want to read it again:

```
POINT #1, SECTOR, BYTE
```

It's that easy. If you keep the values in a table, you can read each record whenever you want. Let's look at a program that will NOTE the locations of each record in an already existing file. Type it in and try it on the file that we created last month with program two.

```
5 REM PROGRAM 3
10 DIM A$(111),TABLE(10,2):TRAP 100
20 ?"Which file to read";:INPUT FILE$
30 RECORD=1:OPEN #1,4,0,FILE$
40 NOTE #1,SECTOR,BYTE
50 TABLE(RECORD,1)=SECTOR
60 TABLE(RECORD,2)=BYTE
70 INPUT #1,A$:?A$
80 RECORD=RECORD+1
90 GOTO 40
100 CLOSE #1:REM END OF FILE
```

Now we know the sector and byte locations of each record. Reading each one will now be a simple matter of looking up its location in TABLE:

```
110 TRAP 40000:?"Which record of ";FILE$;" to read";:INPUT
RECORD
120 SECTOR=TABLE(RECORD,1)
130 BYTE=TABLE(RECORD,2)
140 OPEN #1,4,0,FILE$
150 POINT #1,SECTOR,BYTE
150 INPUT A$:? A$
160 CLOSE #1
170 GOTO 110
```

Simple! We used a TRAP 40000 to clear the trap to line 100- if something goes wrong, we want to know about it.

Now let's write a program that will:

1) Input stuff from the keyboard
2) Write it to a disk file
3) Re-read and edit any record
4) Edit any record in a file by running a routine like program 3 to find it and then writing over the old record using POINT.

To do this we're going to have to use a couple of tricks. First, in order to make re-reading and editing individual records easier, we have to ensure that all records are the same length. If some records were only a few characters long, we couldn't edit them to be any longer. Let's use 35 characters, a little less than the width of the screen.

Instead of INPUTing from the keyboard, we'll write our own little key-in routine that will demonstrate another use of the OPEN and CLOSE commands and make the program look much more professional.

Now that we've got a file that is composed of 35-character records, how can we edit it? Well, if we NOTE the location of the file pointer right before we PRINT each record to the file, we'll know where to POINT to re-read that particular record. But once we save the file, our pointers will be lost. To find them again, we will use program three.

For simplicity's sake, we'll just have the user re-type the line to be edited instead of using a complicated editing routine like a word processor might.

Subroutines:

Lines 100- 140: Decide whether to create a file and then edit it or to get the pointers for an existing file and edit that.

Lines 200-260 A fancy little key-in routine that decides what operation to perform and calls the appropriate subroutine. Time to explain that mysterious little OPEN statement in line 10. It opens IOCB #1 to the keyboard, so when we say "GET #1, COMMAND it puts the ATASCII value of the key pressed into COMMAND. Very useful.

Lines 300- 350: Reads the current file to the screen. Depending on which flags have been set, it may print the line numners, LPRINT, or just print the lines.

Lines 600- 680: Edit any record by looking up its location in the table and printing over it.

Lines 800- 860: Create a file. Uses the subroutine at line 1000.

Lines 1000- 1090: The line- entering routine. This helps make the program idiot-proof, and prevents the user from entering control characters in the lines. We could just replace this whole routine with INPUT LINE$ and the program would still work, but this is more intersting.

Lines 2000- 2040: Find the pointers for the file.

Now that we have the capability to create, read, and edit stuff on disk, think of the applications. Just about anything can be stored permanently on disk- have a game keep a permanent record of the top 10 high scores- you could write a gambling simulation that keeps a permanent record of your bankroll. Or a list of your freinds' phone numbers...

```
10 DIM TABLE(300,2),LINE$(35),FILE$(20):OPEN #1,4,0,"K":POKE
82,0
100 REM *** WHERE TO? ***
110 CLOSE #2:POKE 702,64:? "Edit an existing file?":GET
#1,COMMAND:? CHR$(COMMAND)
120 IF CHR$(COMMAND)="Y" THEN ? "Filename:";:INPUT
FILE$:GOSUB 2000:GOTO 200
130 IF CHR$(COMMAND)<>"N" THEN GOTO 110
140 GOSUB 800
200 REM *** EDIT THE FILE ***
205 REM -GET A COMMAND-
210 ? :? "File: ";FILE$:? "(R)ead (E)dit (L)ist (P)rint
(Q)uit ":?
```

```
220 POKE 702,64:GET #1,COMMAND
225 IF CHR$(COMMAND)="Q" THEN RUN
230 IF CHR$(COMMAND)="E" THEN GOSUB 600
240 IF CHR$(COMMAND)="R" THEN GOSUB 300
250 IF CHR$(COMMAND)="L" THEN LFLAG=1:GOSUB 300
255 IF CHR$(COMMAND)="P" THEN PFLAG=1:GOSUB 300
260 GOTO 210
300 REM -READ FILE TO SCREEN-
310 CLOSE #2:OPEN #2,4,0,FILE$:TRAP 350:LNUM=1
320 IF LFLAG THEN ? LNUM;") ";:LNUM=LNUM+1
330 INPUT #2;LINE$:? LINE$:IF PFLAG THEN LPRINT LINE$
340 GOTO 320
350 LFLAG=0:PFLAG=0:A=0:CLOSE #2:RETURN
600 TRAP 680:? "Edit which record";:INPUT RECORD
610 CLOSE #2:OPEN #2,4,0,FILE$
620      POINT      #2,TABLE(RECORD,1),TABLE(RECORD,2):INPUT
#2,LINE$:? :? RECORD;")":? LINE$
630 ? "Re-type line:":FLAG=-1:GOSUB 1000:FLAG=0
640 CLOSE #2:OPEN #2,12,0,FILE$
650 POINT #2,TABLE(RECORD,1),TABLE(RECORD,2):?
670 ? #2;LINE$:GOTO 200
680 CLOSE #2:GOTO 200
800 REM *** CREATE FILE ***
820 ? :? :? "Create a file":? :?  "Use what filename";:INPUT
FILE$:OPEN #2,8,0,FILE$:LNUM=0:? "  35 characters per  line.
";
825 ? "Put an '2' at...the  start of  a new  line to  end.
The.....file will be written to the disk as.....you type."
827 REM PERIODS IN ABOVE LINE ARE
828 REM SPACES
830 LNUM=LNUM+1:? :? "Line ";LNUM:COL=0
840 GOSUB 1000:IF DONE THEN RETURN
850                                                    NOTE
#2,SECTOR,BYTE:TABLE(LNUM,1)=SECTOR:TABLE(LNUM,2)=BYTE
860 ? #2;LINE$:GOTO 830
1000 REM *** TYPE A LINE OF TEXT ***
1010    GET     #1,A:IF     A=126     AND     COL>0     THEN
LINE$(COL,COL)="":COL=COL-1:? CHR$(A);:GOTO 1000
1020 IF A=155 THEN GOTO 1070
1030 IF A=ASC("2") AND COL=0 THEN DONE=1:RETURN
1040 IF (A>27 AND A<32)  OR (A>124 AND A<128) OR  (A>155 AND
A<160) OR A>253 THEN ? "[BUZZER]";:GOTO 1000
1050 COL=COL+1:LINE$(COL,COL)=CHR$(A):?  CHR$(A);:IF  COL=35
THEN 1070
1060 GOTO 1000
1070 IF COL<>35 THEN FOR LOOP=COL+1 TO 35:LINE$(LOOP,LOOP)="
":NEXT LOOP
1080 COL=0:RETURN
1090 IF FLAG=-1 THEN RETURN
2000 REM **** GET FILE POINTERS ***
2010 CLOSE #2:OPEN #2,4,0,FILE$:COUNT=1:TRAP 2040
2020                                                   NOTE
#2,SECTOR,BYTE:TABLE(COUNT,1)=SECTOR:TABLE(COUNT,2)=BYTE
2030 INPUT #2,LINE$:COUNT=COUNT+1:GOTO 2020
2040 RETURN
```

*** CHRIS CRAWFORD ***

ASSEMBLY LANGUAGE COURSE

LESSON FIVE:
INDEX REGISTERS & LOOPING

We are now going to expand the model of the 6502 that you have been using. Until now, the 6502 I have described had nothing more than a status register, program counter, and accumulator. Now I am going to reveal the existence of two new registers in the 6502: the X- and Y-registers.

These two registers are eight-bit registers just like the accumulator. You can load numbers into them and store them out just as you can with the accumulator. You cannot do arithmetic or Boolean operations with them as you can with the accumulator. But you can do a number of very special things that greatly increase the power of the 6502.

Let's start with the simple move instructions. The first are LDX and LDY, which load the X- and Y-registers the same way that LDA loads the accumulator. Then there are STX and STY, which store the X- and Y-registers the same way that STA stores the accumulator. There are also four commands for transferring bytes between registers; these are TAX (transfer A to X), TAY (transfer A to Y), TXA (transfer X to A), and TYA (transfer Y to A).

Then there are four special instructions that you will use very often. These are INX and INY, which increment (add one to) the X- and Y-registers, and DEX and DEY, which decrement (subtract one from) the X-and Y-registers.

Finally, we have the CPX and CPY commands, which compare X or Y with the operand of the instruction. These two instructions operate in exactly the same way that the CMP instruction works, except that they use the X- and Y-registers instead of the accumulator.

What are these two registers used for? Well, they are sometimes used as temporary registers. If you are in the middle of a lengthy computation, and you need to save a value currently in the accumulator to make room for something else, the X- and Y-registers are a handy place to stuff values away for temporary storage. Programmers do this all the time.

However, temporary storage is not the real purpose and vaue arise from their utility as index registers. Index registers go hand in hand with loops; the best way to show you how they are used is to dump the whole schmeer at once and then explain it.

So consider the following problem: your program has to deal with the possibility of user errors. Suppose you require the user ty type in a file name for your program to read. What happens if this file is not on the disk? You have to put an error message on the screen that says, "FILE NOT ON DISK!" How do you print the message? Here's a sample bit of code that will do it:

```
LDX   #(ENDMSG-ERRMSG-1)
LOOP1 LDA   ERRMSG,X
STA   SCREEN,X
SEC
SBC   #$20
DEX
BPL   LOOP1
JMP   ELSWHR
ERRMSG DB   'FILE NOT ON DISK1'
ENDMSG DS   1
```

Let's take apart this code and explain it step by step. First thing we do is load the X-register with the number of characters (minutes one) in the message. The expression (ENDMSG-ERRMSG-1) will calculate that length at assembly time. This turns out to be 17 characters long. If we were pedestrian about it we could have just written LDX #16, but this way, if we decide to change the message we don't have to remember to go back and change the LDX command. Neat, huh?

OK, so now we have a 16 in the X-register. Now the 6502 comes to the next command -- LDA, ERRMSG,X. This command tells it to load the accumulator with the byte at (address ERRMSG, indexed by X). What this means is as follows: the 6502 will take the address ERRMSG and add the value of the x-register to that address. It will then go to the address so calculated and load the accumulator with the contents of that address. Since X contains a 16, the 6502 will go to the 16th byte after the first byte in the table ERRMSG. If you count characters, you will see that the 16th byte is the exclamation point. So the 6502 will load the ASCII code for an exclamation point into the accumulator.

The next two instructions (SEC, SBC #$20) are necessary to correct for the Atari's nonstandard handling of ASCII codes. They make sure that the exclamation will be printed on the screen as an exclamation point.

The next instruction (STA SCREEN,X) stores the reslt indexed by X. The 6502 will add the contents of X (still 16) to the address SCREEN. It will then store the contents of the accumulator into that address. If that address is part of screen RAM, then you will see an exclamation point appear on the screen.

The next instruction that the 6502 encounters is the DEX instruction. This instruction subtracts one from the X-register, making it a 15.

Next, the 6502 comes to the instruction BPL, LOOP 1. This will branch if the N-flag is clear. The vaue of the N-flag is affected by a DEX instruction. The value of bit D7 of the result is transferred to the N-flag. Bit Dd7 of 15 is a zero, hence the N-flag is clear, hence the 6502 will indeed take the branch. Note that it branches back up to LOOP 1.

Now it will repeat the process, only this time X contains a 15, not a 16. It will therefore grab the 15th character, an ASCII 'K', and store that to the screen position just before the exclamation point. Then it will subtract one form X to get a 14, and will continue the loop.

This process will continue, with the 6502 grabbing bytes in reverse order form the table and storing them onto the screen, until after the 6502 does the seroth byte. When X contains a zero, and the 6502 executes a DEX, it obtains the result $FF. This sets the N-flag. When the 6502 encounters the BPL command, it will NOT take the branch; instead, it will skip the branch and go on to the JMP statement. The loop is terminated.

In this one fragment of code you have seen two major ideas: indexed addressing and looping. They are so closely related that it is hard to talk about one without talking about the other.

You can use indexed addressing with either the X-register or the Y-register. You most commonly use indexed addressing with the LDA and STA commands, but you can also use it with many of the other 6502 commands: ADC, SBC, CMP, AND, ORA, EOR, LSR, ROR, ADL, and ROL can all be used with indexed addressing. Indexed addressing allows you to work with tables or arrays of data.

There is one ugly catch: all of your arrays must be less than 257 bytes long, because the index registers are only eight bits wide. Most of the time this is not a serious problem. However, if you must address a larger table or array, you can use indirect addressing. To do this, you calculate the address that you desire to access, store that address in two contiguous bytes on page zero (low, then high) -- we call these two bytes a pointer -- and then refer to the pointer like so:

```
LDA   (POINTER), Y
```

This instruction will take the address out of pointer, add the value of Y to it, and load the accumulator with the contents of the address so calculated. If POINTER contains $4567 and Y contains a 2, then the 6502 will load the acumulator with the contents of address $4569. You are still restricted by the size of Y, but you can always go back and change the POINTER if you need to span larger arrays. In this case, you frequently just leave Y equal to zero and do all of your indexing directly with changes to POINTER.

The last topic I will take up is termination techniques. Every loop must somehow be terminated if you are to avoid the problem of the Sorcerer's Apprentice. You will note that the programming example I gave used a rather odd approach. I started at the end of the array and worked backwards. Why not start at the beginning and work forwards? It's slightly more efficient going backwards than forwards. When you go forwards, you have to terminate the loop with:

```
INX
CPX   #17
BNE   LOOP1
```

Whereas when you go backwards, you need only use:

```
DEX
BPL   LOOP1
```

Going backwards you save one instruction. However, if this confuses you, feel free to count forward; that works, too, only it's a little less efficient.

There is also a problem on choosing whether to BNE or BPL. BPL restricts you to a range of only 127 bytes, but BNE, but index from ERRMSG-1 and SCREEN-1 instead of ERRMSG and SCREEN.

There are lots of other sneaky ways to terminate loops, but they fall into advanced topics.

MC68000
MOTOROLA'S
16/32 BIT MICROPROCESSOR

by SIG-ATARI's Tim Barr

reprinted from MACE

[Attention all you hackers out there! Are you anxious to get at the new ST Line of Atari Computers? Here's a little background info about the heart and soul of these new computers. This article was taken from the SIG Atari section of CompuServe.]

The MC68000 is a 16-bit MPU with 17 general purpose 32-bit registers, a 32-bit program counter and a 16-bit status register. The first eight GP registers (D0-D7) are used as data registers. The next seven GP registers (A0-A6) and the system stack pointers can be utilized as address registers and pointers for software use. The data registers can be used for different data sizes. These sizes are: BYTE (8-bit), WORD (16-bit), and LONG WORD (32-bit) operations. The MPU has a 24-bit address bus (actually it is 23-bit address line and a HI BYTE and LOW BYTE select lines which gives you 8 megawords or 16 megabytes of memory) and a 16-bit data bus.

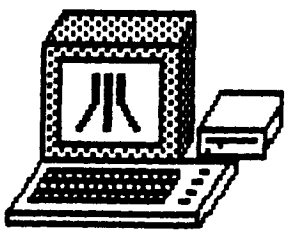
Five basic data types are supported by the instruction set. These data types are: BITS, BCD DIGITS (4-bits), BYTES (8-bits), WORDS (16-bits), and LONG WORDS (32-bits). The MPU has 14 address modes of six basic types: REGISTER DIRECT, REGISTER INDIRECT, ABSOLUTE, PROGRAM COUNTER RELATIVE, IMMEDIATE, and IMPLIED. The surprising thing about the 68000 is that it only has 56 instruction types and a total of only 88 actual instructions. The actual 16-bit OP-CODE that the system uses is a combination of an instruction and an addressing mode, GP register number, an OP-MODE, instruction specific data, or any combination of the four. (Instruction specific data is such information as shift direction, branch conditions, operation size, etc.) This provides you with over 1000 acutal instructions, but keeps the total number of instructions small.

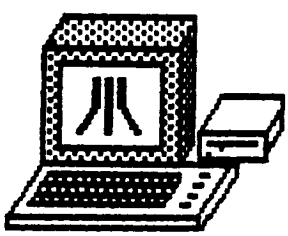
I will try to upload a list of the basic instructions of the 68000 [to CompuServe], but I wanted to mention a few of them here. The MPU can perform add and subtract functions on BCD digits in groups of two digits. It can also multiply and divide binary numbers in these formats: 32-bit / 16-bit => 32-bit and 16-bit * 16-bit => 32-bit. The signed divide is the slowest instruction to execute on the 68000. It takes 170 clock cycles or 21.25 microsec. at a clock speed of 8 MHZ. This means that you could divide any 32-bit number in memory by any 16-bit number in memory over 47,000 times in ONE SECOND!!

I hope that this has given you a general idea of the power of the processor that the new Ataris will be using. If you are interested in more information, I would like to suggest a couple of books that are available. The first is published by Howard K. Sams & Co. and is called "68000: PRINCIPLES AND PROGRAMMING" by Leo Scanion. You should be able to order this book through the Howard Sams section of CompuServe (GO SAM). The second book I would like to recommend is "M68000 PROGRAMMERS REFERENCE MANUAL" available from:

Motorola Semiconductor Products Inc.
P.O. Box 20912
Phoenix, AZ 85036

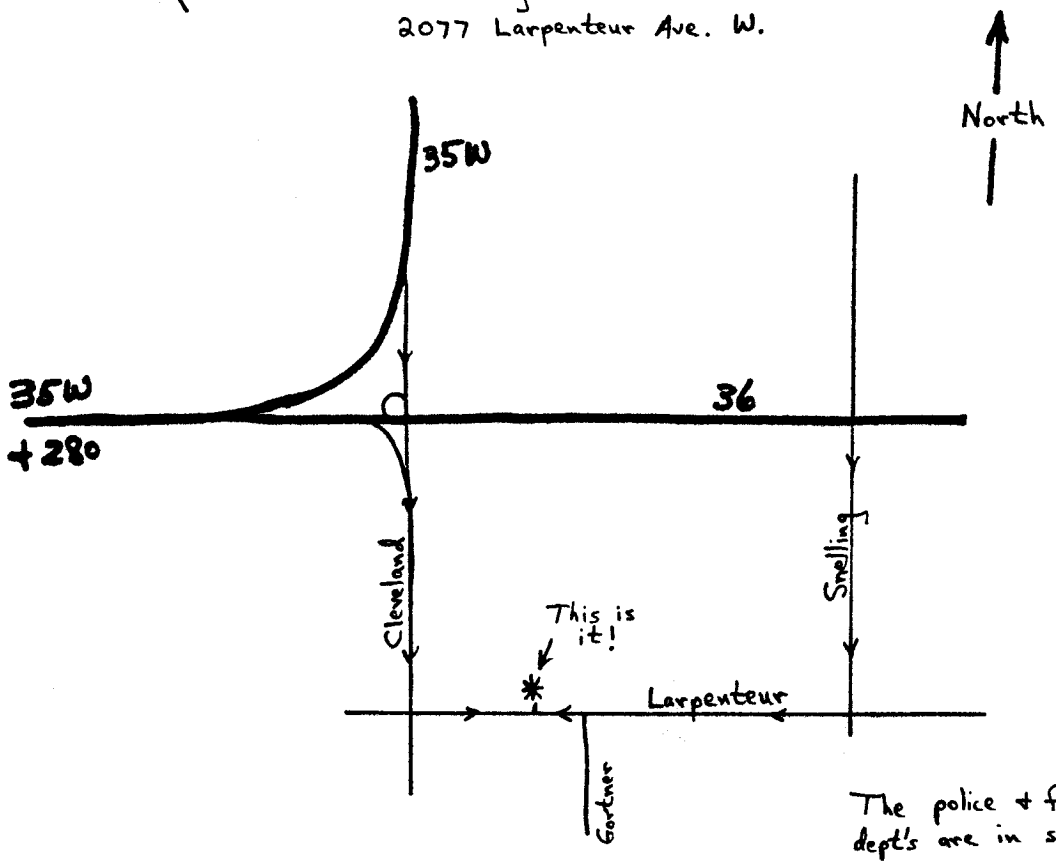
Ask for document # M68000UM(AD4).





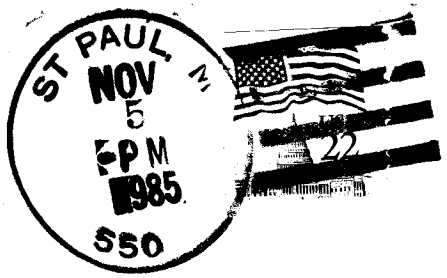
The NEW SPACE Meeting Place

Falcon Heights Comm. Center
2077 Larpenteur Ave. W.



The police + fire dept's are in same bldg as our meeting.

St. Paul ATARI Computer Enthusiasts
2589 Fisk St.
Roseville, MN 55113



J. P. Scheib
3944 24th Av. S.
Minneapolis, Mn. 55406