1982

Poker (1.0) Programmers Guide

Lawrence Snyder

Report Number:

83-434

The Poker (1.0) Programmers Guide

Lawrence Snyder

ABSTRACT

The Poker Parallel Programming Environment is a graphics-based interactive system for writing and running CHiP programs. The programs can be emulated or run on the Pringle (when completed). Poker runs on the VAX 11/780 under UNIX using two displays (see Figure 1). Poker permits the programmer to encode a parallel algorithm in a convenient, "high level" interactive environment, but because our approach is somewhat nonstandard, we begin with a discussion of our view of the parallel programming activity. The sections of this document are:

I. CHiP programming is something else

II. Poker Programmer's Reference Guide

Comments on this document or the programs to which it refers are eagerly solicited.

CSD-TR-434

20 December 1982

This work is part of the Blue CHiP Project. It is supported in part by the Office of Naval Research Contracts N00014-80-K-0816 and N00014-81-K-0360. The latter is Task SRO-100.

Acknowledgements

The Poker System is the product of the ideas and effort of many people. Janice E. Cuny and Dennis B. Gannon, in addition to contributing to the definition of the XX programming language, were a continual source of ideas, judgement and constructive criticism. Christopher A. Kent contributed extensively to the overall design as well as the programming. Version 1.0 of Poker was written during the summer of 1982 by a delightful and committed group of gentlemen, the "poker players": Steven S. Albert, Carl W. Amport, Brian G. Beuning, Alan J. Chester, John P. Guaragno, Christopher A. Kent, John Thomas Love, Eugene J. Shekita, and Carleton A. Smith. Concurrently, the coordination phase of Poker was written under the direction of Janice E. Cuny by Karen L. Pickering and Ellen F. Scanlon. J. Timothy Field and Alejandro A. Kapauan cheerfully explained the details of the Pringle architecture. Julie K. Hanover expertly prepared the Poker documents under tight time constraints. J. Timothy Korb and Robert L. Brown gave helpful guidance on the Bit-Graph, and Bob wrote the interfacing software. Vance Waddle suggested the name, after Poker's "peeking and poking" trace facilities. The contributions of all of these people are deeply appreciated.

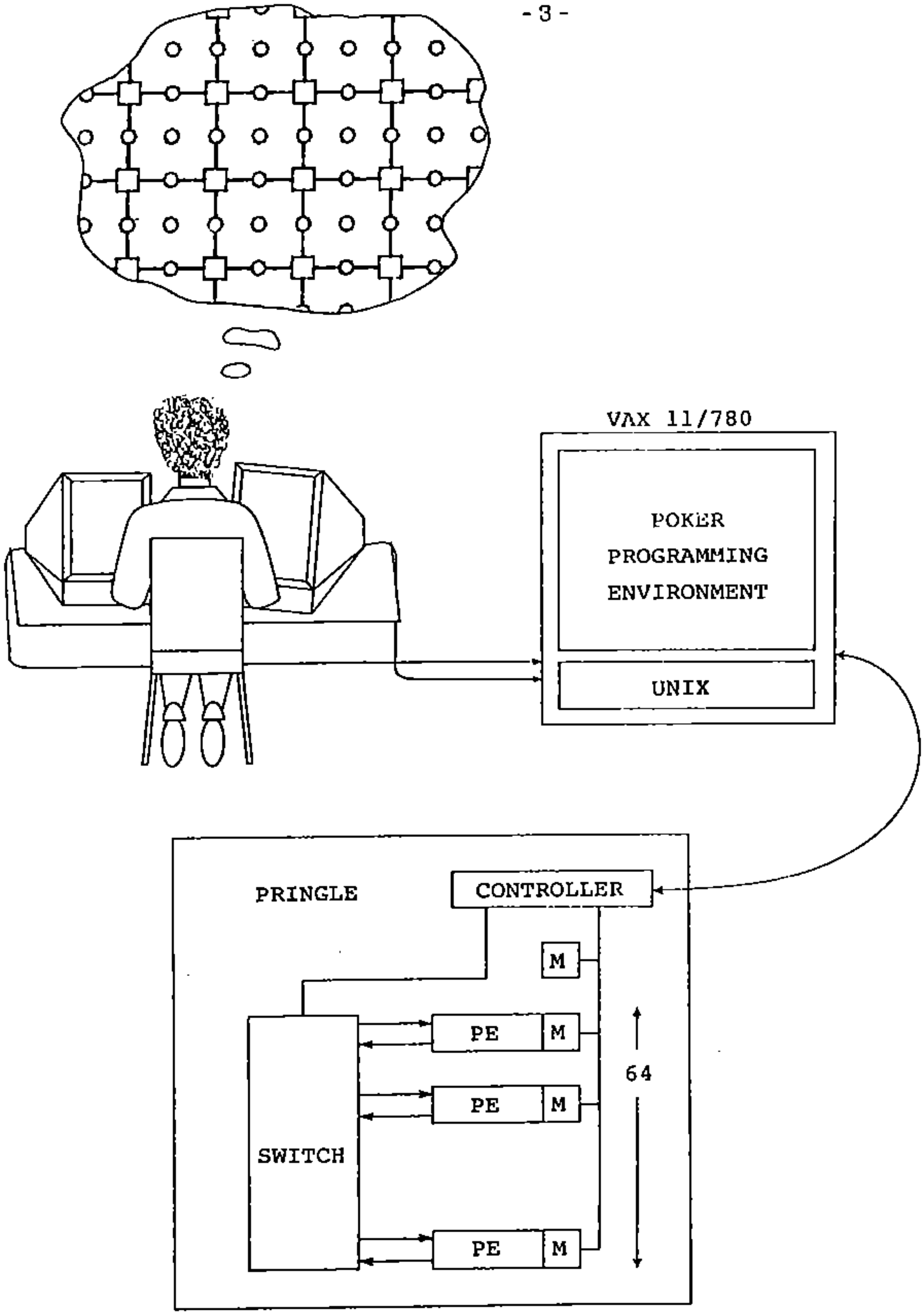


Figure 1. The Structure of the Poker Programming Environment.

I. CHiP Programming is Something Else

The programming environment provided by Poker is somewhat unconventional due partly to novel properties of the CHiP Computer and partly to novel properties of the system itself. To increase the accessibility of subsequent sections, we discuss here the *activity* of CHiP programming and the *role* Poker plays.

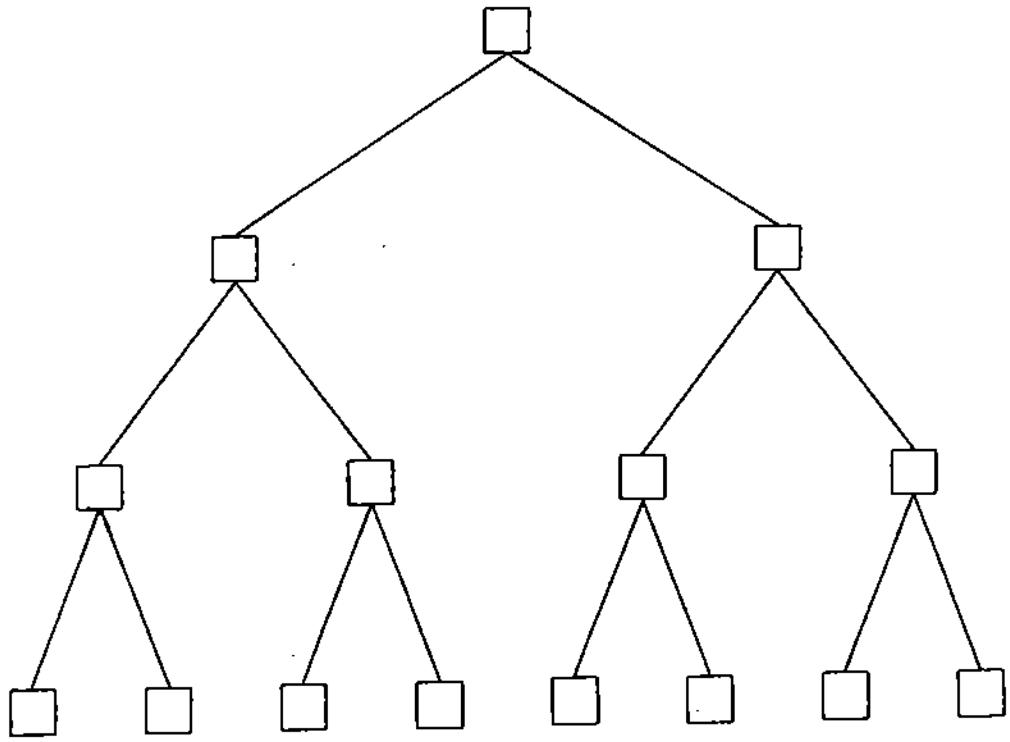
Programming, of course, is the conversion of an (abstract) algorithm that is "machine independent" into a form suitable for execution on a particular computer. Thus, to begin programming a CHiP machine, we need to have a parallel algorithm in mind. The algorithm is presumed to have the form of a graph whose vertices are processes and whose edges specify the communication paths among the processes.

For example, Figure 2 gives an algorithm that uses a binary tree as the communication graph. The algorithm finds the maximum of a set of numbers (stored one per process in a local variable called "val") and then multiplies each number by the maximum. The maximum is found by "floating" the largest value in each subtree to the root of that subtree. Then the global maximum is broadcast back through the tree where each process multiplies it times its local "val." Notice that although there are fifteen processes in the tree, there are only three *types* of processes used.

The conversion of this algorithm to run on a CHiP computer, i.e., the programming, is straight forward.* It involves

---

*Assuming familiarity with the CHiP Computer. Complete information can be found in "Introduction to the Configurable, Highly Parallel Computer," Lawrence Snyder, *Computer*, 15(1): 47-56, January 1982.



leaf process:
write val to parent;
read max from parent;
val ← val · max;

ancestor process:
read x from left child;
read y from right child;
write max (x, y, val) to parent;
read max from parent;
write max to left child;
write max to right child;
val ← val · max;

root process:
read x from left child;
read y from right child;
max ← max (x, y, val);
write max to left child;
write max to right child;
val ← val · max;

Figure 2. An algorithm; each leaf is an instance of the leaf process, the root is an instance of the root process and all other nodes are instances of the ancestor process.

(a) embedding the communication graph into the switch lattice,
(b) programming the process types in a sequential programming language,
(c) assigning one of the process types to each processor,
(d) naming the data path ports, and
(e) compiling, assembling, coordinating, and loading the program.

We consider each of these activities in turn.

Embedding the communication graph into the switch lattice requires that we program the switches of the lattice so that the processors have a topology that matches (or is a super set of) the topology of the communication graph. This embedding operation is done graphically (rather than symbolically) in the Poker System using the Switch Settings mode. Figure 3 illustrates a particular embedding of the fifteen node binary tree onto the lattice. Processor (1,2) is the root of the processor tree, processor (1,1) is a leaf, and processor (1,3) is unused.

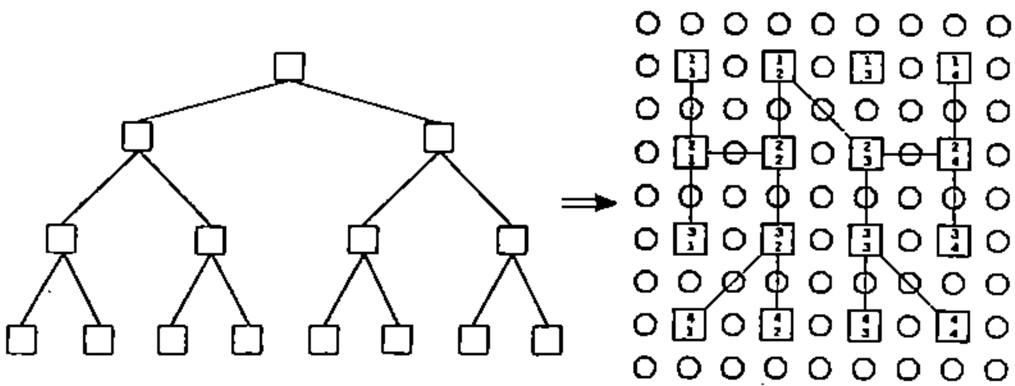


Figure 3: An embedding of the 15 node binary tree.

Next we program the three process types in the sequential language XX. Each process is viewed as a procedure with (optional) parameters and local variables. In addition to the usual declarations we must specify the *port names*, symbolic names used by a process to refer to other processes with which it communicates. Figure 4 shows the XX code for the three process types. In the programs the symbol '<-' is used for input/output; assigning to a port name, e.g., PARENT <- val, causes output and assigning from a port name, e.g., max <- PARENT, causes input.

```
code leaf (val);
ports PARENT;
begin
int max, PARENT;
PARENT <- val;
max <- PARENT;
val:=val * max;
end
```

```
code ancestor (val);
ports PARENT,LCHILD,RCHILD;
begin
int x,y, max, val,
  PARENT, LCHILD, RCHILD;
x <- LCHILD;
y <- RCHILD;
if x>y then max:=x
  else max:=y;
if val > max then max:=val;
PARENT <- max;
max <- PARENT;
LCHILD <- max;
RCHILD <- max;
val:=val * max;
end
```

```
code root (val);
ports LCHILD, RCHILD;
begin
int x,y, max, val,
  LCHILD, RCHILD;
x <- LCHILD;
y <- RCHILD;
if x>y then max:=x
  else max:=y;
if val > max then max:=val;
LCHILD <- max;
RCHILD <- max;
val:=val * max;
end.
```

Figure 4. Code for the three process types.

The construction of the processor tree in the switch lattice to match the communications graph gives an implicit association between the processes of the algorithm and the processors. We make this relationship explicit by assigning process names to the appropriate processors using the Code Names mode of the Poker System. Figure 5 gives the result.

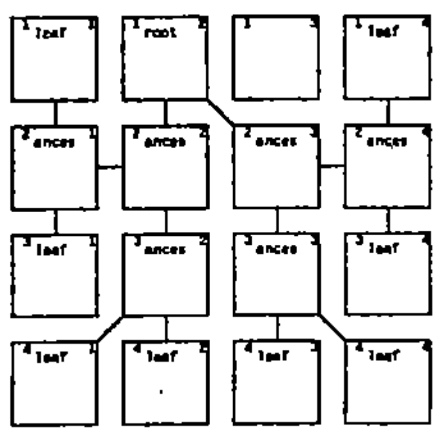


Figure 5. Assignment of process names to processors; note that the name "ancestor" has been clipped to five characters.

Next, the port names mentioned in each process must be associated with a specific data path. Each processor has eight ports corresponding to the compass points. Only those connected by an active data path to another PE need be named. This activity is performed using the Port Names mode of Poker. Figure 6 shows the result of naming the ports.

The algorithm is now programmed. Next, each process type mentioned in the Code Names specification is compiled into assembly code. The assembly code is then "coordinated," i.e., modified so that the CHiP Computer can run it synchronously. The coordinated programs are assembled to produce processor object code. The interconnection structure is "compiled" to produce switch object code. The object codes are loaded into the machine and executed.

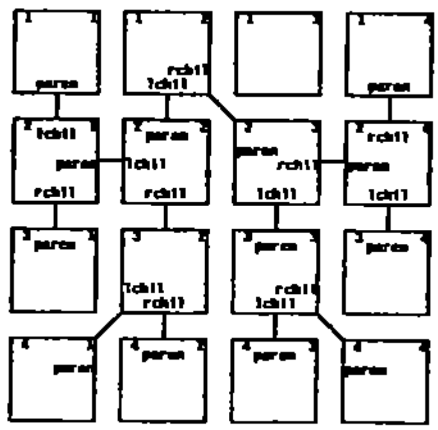


Figure 6. The specification of the port names; note that the names have been clipped to the first five characters.

II. Poker Programmer's Reference Guide

This section gives a succinct description of the facilities available to the programmer with the Poker Programming Environment. The emphasis is on "what can be done" rather than "how to achieve particular results." Although the sections are self-contained, and can be referred to independently, it is suggested that the reader peruse the sections sequentially first. The sections are:

1. The facilities and the display
2. Cursor motions
3. CHiP parameters mode
4. Switch settings mode
5. Port names mode
6. Code names mode
7. The XX programming language
8. Command request mode
9. Trace values mode
10. Port values mode

A. Catastrophic Bugs
B. Summary of Key Definitions

Additional information is available in "Introduction to the Poker Programming Environment," Lawrence Snyder, Purdue University Technical Report CSD-TR-433, 1983.

To access the Poker System (from the Research VAX) the user should include the directory "/usr/lxs/poker/bin" in his search path. This requires a (one-time) change to the PATH line of your .profile file. The required modification is to append the text ":/usr/lxs/poker/bin" to the PATH line.

1. The facilities and the display

The Poker System uses two displays: a BBN BitGraph Display and a conventional character display (e.g., ADDS Regent 40).* The user should

*It is possible, though inconvenient, to use just the BitGraph.

be logged into both terminals and should have both referring to a common directory. [To avoid name conflicts, it is advised that the directory be clear (initially).]

The command 'poker' from the BitGraph terminal causes the system to be entered. Thereafter, the display will have a form of the type shown in Figure 1. Below the horizontal line is the "field" in which most activity takes place. The field changes depending on how the programming environment is being modified. Above the line is the status information. The "lattice" gives a schematic picture of the processing elements (PEs) of the machine being programmed. A box circumscribes that portion of the lattice displayed in the field giving the user geometric context. The chalkboard gives status information that is largely self explanatory. The last line of the chalkboard is where all diagnostics are printed. The command line is used to give commands (naturally), to present textual parameters, and to perform certain kinds of editing. Poker execution always begins in the CHiP parameters mode.

The Poker system is interactive: *virtually all key strokes cause an immediate action.* (Exceptions to this statement are described below.) All actions, except text insertion and some cursor motions, are composit key strokes formed either by *simultaneously striking* the control key and a letter key (e.g., we write ^h to denote simultaneously striking the control key and the letter **h** (which causes the cursor to backspace)), or by first striking the escape key (written *esc)* followed by the simultaneous striking of the control key and a letter (e.g., esc-^a is the command to abort and return to UNIX). Should *esc* be inadvertently struck, it can be cleared by striking *esc* again.

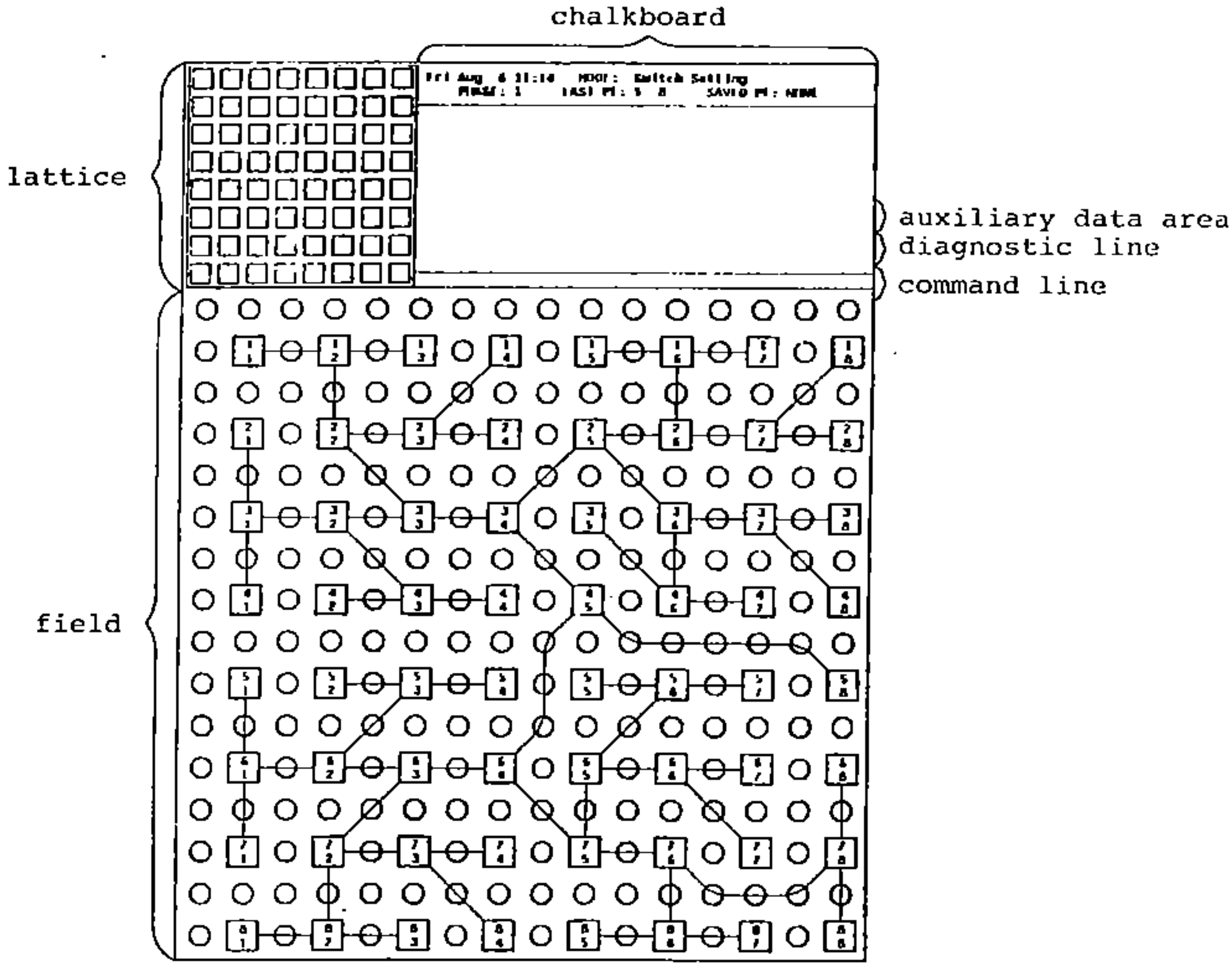


Figure 1.

2. Cursor Motions

Movement around the lattice and within the PEs is controlled by the positive numeric keys of the key pad (located on the right side of the keyboard and illustrated in Figure 2). Two kinds of motions are provided: gross cursor motions and fine cursor motions. The gross cursor motions, which are two-key operations composed of an esc followed by a directional key, usually move to the next PE in the indicated direction. Fine motions, which are given just by a directional key, vary in meaning with the mode.

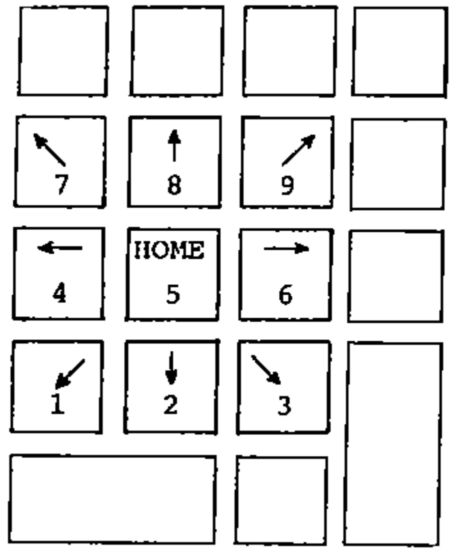


Figure 2. Meaning of the key pad keys.

Fine Moves	Directions	Gross Moves
4	WEST	esc-4
7	NORTH-WEST	esc-7
8	NORTH	esc-8
9	NORTH-EAST	esc-9
6	EAST	esc-6
3	SOUTH-EAST	esc-3
2	SOUTH	esc-2
1	SOUTH-WEST	esc-1
5	HOME	esc-5

Figure 3. Gross and fine cursor motions.

3. CHiP Parameters

Purpose: To specify the characteristics of the CHiP machine being programmed.

Display: The current values of the CHiP computer's parameters are given in the command line; their meaning is described in Figure 4.

Activity: The cursor is moved right and left along the command line using (gross or fine) east and west cursor motions. Numbers entered replace the symbol pointed to by the cursor. The new values take effect when the mode is changed provided they are in range and satisfy the constraints; no changes take place if *any* parameter is illegal.

Limitations: Specification of $n=64$ is not currently possible due to inadequate page table space in the UNIX kernel; $p>1$ is not fully implemented.

Parameter	Range	Constraints	Default
n - size, number of PEs on the side of the lattice	$2 \leq n \leq 64$	$n=2^k$	8
w - internal corridor width, the number of switches separating two adjacent PEs	$1 \leq w \leq 4$		1
u - external corridor width, the number of switches between the perimeter and the edge PEs	$1 \leq u \leq 4$	$u \leq w$	1
d - degree, number of data paths incident to PEs and switches	8	fixed	8
c - crossover level, number of distinct data paths through a switch	$1 \leq c \leq 4$		2
p - number of phases, the size of of the switch memory	$1 \leq p \leq 16$		1

Figure 4. Description of the CHiP Parameters.

Change n: If the value of n is increased, the old lattice becomes the upper left-hand corner of the new lattice; if n decreases, the new lattice is the upper left-hand corner of the old lattice.

Change w: A change in w causes switch columns (rows) to be added or removed from the right (bottom) of vertical (horizontal) switch corridors. Existing switches retain their settings; new switches are unset.

Change u: A change in u causes switches to be added or removed at the perimeter. Existing switches retain their settings; new switches are unset.

Change c: A change in c permits the number of distinct data paths through a switch that *can be set* to be either increased or decreased.

Change p: If p is increased, phases with consecutive higher numbers are added; if p is decreased, phases with higher indexes are removed. Added phases are clear.

Recognized keys:

esc-^a abort, return to UNIX without saving state.
esc-^e exit, return to UNIX and save CHiP parameters, switch settings, port and code names in the current directory.
esc-^l redraw; the screen is redrawn.
esc-^o output screen; the BitGraph's raster memory is dumped to a file named BG*xxxxxx* in the current directory, where *xxxxxx* is a random number
<mode> change state to reflect revised parameters, if legal, and switch to a new mode according as mode is

esc-^p port names mode
esc-^d code names mode
esc-^w switch settings mode
esc-^r command request mode
esc-^v port values mode
esc-^t trace values mode

<text> replaces the symbol at the cursor

4. Switch Settings

Purpose: To specify or modify a processor interconnection structure for the lattice.

Display: The current processor interconnection structure of (a portion of) the lattice for this phase is shown in the field; boxes represent processors, and circles represent switches.

Cursor motion: Gross cursor motions advance the cursor to the next PE in the indicated direction; fine cursor motions advance the cursor to the next entity (PE or switch) in the indicated direction. 'Home', from a switch causes the cursor to return to Last PE, from a PE causes it to go to the command line, and from the command line to go to the Last PE.

Activity: The cursor is moved around the lattice. If the insert mode is set, a wire is "pulled along" from the current position to the cursor's new position. If the delete mode is set, wires followed by the cursor are removed. At a switch all wires common to the current level are highlighted, (with bold strokes). If the chase mode is set, the cursor follows the wire in the direction indicated until it reaches a PE, or terminates, or reaches a switch that fans out, or cycles.

Recognized keys:

esc-^a abort, return to UNIX without saving state.
esc-^e exit, return to UNIX and save the current values of the CHiP parameters, the switch settings and the code and port names.
esc-^l redraw; the screen is redrawn.
esc-^o output screen; the BitGraph's raster memory is dumped to a file named BG*xxxxxx* in the current directory, where *xxxxxx* is a random number.
<mode> switch to the indicated mode:

- esc-^c CHiP parameters mode
- esc-^p port names mode
- esc-^d code names mode
- esc-^r command request mode
- esc-^v port values mode
- esc-^t trace values mode

<text> is placed on the command line.
^h backspace; if the cursor is on the command line.
^c center the display so that the PE whose index is given on the command line is as close to the center of the field as possible consistent with the requirement that the field remain fully utilized; if the command line is blank, use the Last PE for centering.
^i insert mode is set, so subsequent cursor motions cause a line to be drawn. From the command line, ^i reads in a switch setting file whose name is given on the command line, or, if none is given, the Switch Set file of the current directory.
^d delete mode is set, so subsequent cursor motions that follow a line cause it to be removed. From the command line, ^d deletes all switch settings.
^x set chase mode, so that (only) the next cursor motion will

follow the line in the indicated direction until it terminates, reaches a PE, reaches a switch that fans out or cycles.

^e end the current mode, i.e., cancel insert, delete or chase.

^l level change; the level of the switch pointed to by the cursor is changed to the next lower level. Repeated use of this key cycles through all assigned levels and one unassigned level.

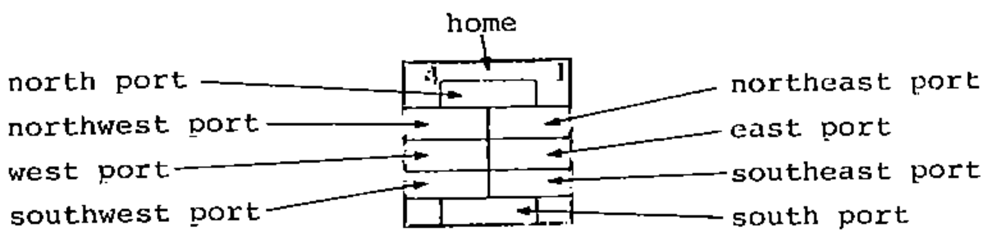
^w writes the current state of the switch settings to a file, SwitchSet, in the current directory.

^p phase change; the phase number, given on the command line, becomes the new phase. [Not fully implemented.]

5. Port Names

Purpose: To specify or modify the names assigned to the eight input/output ports of a PE.

Display: The current port names of (a portion of) the lattice for this phase are shown in the field. The display format shows one box representing the PEs; the other display format shows boxes representing the PEs and lines representing the interconnection structure; a key (~t) toggles between the two. Names of up to 16 characters, clipped to the first five characters, are shown in the PE boxes:



Cursor Motion: Gross cursor motions advance the cursor to the home position of the next PE in the indicated direction; fine cursor motions move the cursor to the first position in the window for the port name for that direction. 'Home', from a port window moves to the home position of this PE, from the home position in a PE to the command line, and from the command line to the home position of Last PE.

Activity: Port names are entered into the appropriate windows to name the ports connecting to the incident data paths. Port names can be any legal identifier of the XX programming language not containing blanks.

Buffering: The port names of any PE can be saved in a buffer (using ~b) that is then displayed in the chalkboard. The saved port names can be deposited into one or more PEs by specifying recipient PE(s) on the command line followed by an insertion (~i). Recipient PE(s) are specified either explicitly by an index pair (i j), or implicitly by an expression where each index position is an index, a relation (<, <=, >, >=) followed by an index, meaning all indices standing in that relation to the index, or a period (.) meaning all index values. Thus a command

. <= 4

followed by ~i causes the first four columns to receive the saved port names.

Recognized keys:

esc-~a abort, return to UNIX without saving state.
esc-~e exit, return to UNIX and save the current values of the

CHiP parameters, switch settings, port and code names.

esc-^l redraw; the screen is redrawn.

esc-^o output the screen; the BitGraph's raster memory is dumped to a file named BG*xxxxxx* in the current directory, where *xxxxxx* is a random number.

<mode> switch to the indicated mode:

esc-^c	CHiP Parameters mode
esc-^w	switch settings mode
esc-^d	code names mode
esc-^r	command request mode
esc-^v	port values mode
esc-^t	trace values mode

<text> if the cursor is in a window, the symbol replaces the symbol pointed to by the cursor; if the cursor is at the home position of a PE or on the command line, the symbol appears on the command line.

^h backspace.

^b buffer the port names of the PE containing the cursor. Modification of the port names of a buffered PE cause it to be removed from the buffer.

^i insert the buffered names into the recipient PE(s). If the command line is blank, the recipient is the PE containing the cursor; if the command line is nonblank, the recipients are given by the command line expression as described in Buffering above.

^d delete port names. If the cursor is in a PE, delete all port names in this PE; if the cursor is on the command line, delete all port names.

^c center the display so that the PE whose index is given on the command line is as close to the center of the field as possible consistent with the requirement that the field remain fully utilized; if the command line is blank, use the Last PE for centering.

^t toggle the display to be in the "other" format; see Display above.

^y display the full (unclipped) entry of the window containing the cursor; the display is given on the auxiliary data line of the chalkboard.

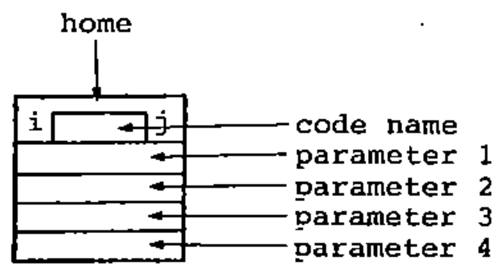
^w write the current values of all port names to the file PortNames in the current directory.

^p phase change; the phase number given on the command line becomes the new phase. [Not fully implemented.]

6. Code Names

Purpose: To specify or modify the names of the XX programs assigned to the PEs or to specify actual parameters to these programs.

Display: The current code names and parameter assignments of (a portion of) the lattice for this phase are given in the field. One display format shows boxes representing the PEs; the other display format shows boxes representing the PEs and lines representing the interconnection structure; a key (~t) toggles between these two. A name of up to 16 characters, clipped to five characters, is shown for the program name, and four symbol strings of up to 16 characters, clipped to ten characters, is shown for the parameters:



Cursor motions: Gross cursor motions advance the cursor to the home position of the next PE in the indicated direction; fine cursor motions (north and south) move to the first position of the windows for the code name and the parameters. Home, from a window moves the cursor to the home position of the PE, from the home position in a PE to the command line, and from the command line to the home position of Last PE.

Activity: Code names and (actual) parameter values are entered into the appropriate positions. Code names can be any legal identifier of the XX programming language not containing blanks, and parameters can be any legal constant of the XX programming language.

Buffering: The code name and parameters of a PE can be saved in a buffer (using ~b) that is then displayed in the chalkboard. The saved values are deposited into one or more PEs by specifying recipient PEs followed by an insertion (~i). Recipient PEs are specified either explicitly by giving an index pair (i j), or implicitly by an expression where each index position is an index, a relation (<, <=, >, >=) followed by an index, meaning all indices standing in that relationship to the index, or a period (.) meaning all index values. Thus, a command

. <= 4

followed by ~i causes the first four columns to receive the saved values.

Recognized keys:

esc-^a	abort, return to UNIX without saving state.
esc-^e	exit, return to UNIX and save the current values of the CHiP parameters, switch settings, port and code names.
esc-^l	redraw; the screen is redrawn.
esc-^o	output the screen; the BitGraph's raster memory is dumped to a file named BG*xxxxxx* in the current directory, where *xxxxxx* is a random number.
<mode>	switch to the indicated mode: esc-^c CHiP parameters mode esc-^w switch settings mode esc-^p port names mode esc-^r command request mode esc-^v port values mode esc-^t trace values mode
<text>	if the cursor is in the window, the symbol replaces the symbol pointed to by the cursor; if the cursor is at the home position of a PE or the command line, the symbol appears on the command line.
^h	backspace.
^b	buffer the code name and parameters of the PE containing the cursor. Modification to any of the entries of the buffered PE cause it to be removed from the buffer.
^i	insert the buffered names into the recipient PE(s). If the command line is blank, the recipient is the PE containing the cursor; if the command line is nonblank the recipient is given by the command line expression as described in Buffering above.
^d	delete port names. If the cursor is in a window, delete the window's entry; if the cursor is at the home position of a PE, delete all entries in the PE; if the cursor is on the command line delete all code names and parameters.
^c	center the display so that the PE whose index is given on the command line is as close to the center of the field as possible consistent with the requirement that the field be fully utilized; if the command line is blank use the Last PE for centering.
^t	toggle the display to the "other" format as described in Display above.
^y	display the full (unclipped) entry of the window containing the cursor; the display is given on the auxiliary data line of the chalkboard.
^w	write the current values of all code names and parameters to the file CodeNames in the current directory.
^p	phase change; the phase number given on the command line becomes the new phase. [Not fully implemented.]

7. The XX Programming Language*

Purpose: The XX (dos equis) programming language is a simplified sequential programming language for defining the codes for processing elements of the CHiP computer.

Activity: Files are created or modified using a conventional UNIX editor. The files are named <name>.x where <name> is the name of a program referred to in the code names entries. For convenience in referring to Poker state information on the BitGraph display, it is recommended that XX program files be developed on the secondary (character) Poker display.

Programs: XX programs begin with a preamble that gives the program name, the formal parameters, trace variables and the port names. The preamble is followed by the program body block:

<program> ::= **code** <id> <parmlist>; <tracelist> <port list> <body>
<parmlist> ::= (<idlist>) | λ
<tracelist> ::= **trace** <idlist>; | λ
<portlist> ::= **ports** <idlist>; | λ
<idlist> ::= <id>, <idlist> | <id>
<body> ::= **begin** <declarations> <statlist> **end**

where the parameters and trace identifiers are limited to a list of at most four identifiers separated by commas and the port id list is limited to a list of 8 identifiers separated by commas. The identifier following **code** names the program and should match the <name> of the file and the <name> used in the Code names entries. The parameters are formal parameters that correspond one-to-one to the actual parameters stored in the Code Names/Parameters entries of the PEs; each formal must be declared in the <declarations> section of the <body>. The trace list identifiers have their values displayed during tracing and they must be declared in the <declarations> section of the <body>. The port list identifiers are the symbolic port names that are assigned physical positions in the Port Names entries, and they must be declared in the <declarations> section of the <body>.

Declarations: There are four data dypes: signed integers (32 bits), signed reals (32 bits), characters (8 bits) and Booleans (1 bit). Except for statement label identifiers, all identifiers, including those appearing in the preamble, must be declared. Simple identifiers are scalar values of the indicated type and identifiers followed by [<unsignint>] are vectors of length <unsignint> of scalar values of the indicated type:

<declarations> := <decl>; <declarations> | λ

*Developed with J. E. Cuny and D. B. Gannon.

<decl> ::= <type> <varlist>
<type> ::= **real** | **int** | **bool** | **char**
<varlist> ::= <varid>, <varlist> | <varid>
<varid> ::= <id> | <id> [<unsignint>]

where no <id> appears more than once.

Statements: The statements are:

<statlist> ::= <lstatement>; <statlist> | <lstatement>
<lstatement> ::= <id>: <statement> | <statement>
<statement> ::= <assignment> | <conditional> |
<while> | <break> | <for> | <compound> | <io>

where <id> is used for tracing rather than the target of **goto**.

Assignment: The Assignment statement is:

<assignment> ::= <varid> := <expression>

where the coercion to the left-hand side identifier type is provided as described in Table 1.

Conditional: In the Conditional statement

<conditional> ::= **if** <expression> **then** <lstatement>
else <lstatement> | **if** <expression>
then <lstatement>

the <expression> must evaluate to a Boolean value and an **else** is associated with the immediately preceding **then**.

While: In the While statement

<while> ::= **while** <expression> **do** <lstatement>

the expression must evaluate to a Boolean value. To assist in synchronization the compiler recognizes the construction **while true do** <lstatement> as a special case and does not generate the conditional branch code.

Break: The Break statement

<break> ::= **break**

has meaning only within the <lstatement> of a While statement, and causes control to skip to the statement following the immediately surrounding While statement.

For: In the For statement

<for> ::= **for** <id> := <expression> **to** <expression> **do**
<lstatement>

the two expressions, the lower and upper limits of the iteration, respectively, are evaluated once prior to beginning the loop. If the lower and upper limits are not integers, they are coerced to integers as described in Table 1.

Compound: Notice that the Compound statement

<compound> ::= **begin** <statlist> **end**

is not a block and may not contain declarations.

I/O: The I/O statements

```
<io> ::= <id> <- <id>
```

are restricted to simple variables, exactly one of which must be a port name. If the port name appears on the right, the statement reads from the indicated port; if the port name appears on the left, the statement writes to the indicated port. Data type consistency is not enforced across the communication links.

Expressions: The expressions

```
<expression> ::= <expression> <binary> <expression> |
    <unary> <expression> |
    <expression> <relational> <expression> |
    <builtin> (<expression>) |
    (<expression>) |
    <unsignint> | <unsignreal> | <character> |
    <boolean>
```

have procedence and association as in the C programming language. Expressions of mixed type are coerced to the higher type, where types are ranked **bool** < **char** < **int** < **real**, as described in Table 1. The operators are given in Table 2.

bool → char: The Boolean bit becomes the least significant bit; others are 0.
char → bool: The least significant bit forms the Boolean.
char → int: The 8 character bits become least significant bits; others are 0.
int → char: The eight least significant bits form the character.
int → real: Converted to floating point notation.
real → int: The floating point value is truncated and converted to integer form.

Table 1. Semantics of representation conversion; conversions not listed are performed transitively: type1 → type2 → type3, etc.

<unary>

+ <real>   no op
- <real>   negation
~ <char>   not

The type indicates the highest type for which the operation is defined; the operation is defined for all lower types.

<binary>

<real> + <real>   addition
<real> - <real>   subtraction
<real> * <real>   multiplication
<real> / <real>   division
<real> **mod** <real>   modulus
<real> >= <real>   greater than or equal
<real> > <real>   greater than
<real> =/ <real>   not equal
<real> < <real>   less than
<real> <= <real>   less than or equal
<real> = <real>   equal
<char> & <char>   and
<char> | <char>   or
<char> || <char>   exclusive or

Table 2. XX operators.

Constants: The constants are unsigned integers and reals in standard formats, quoted (') characters and **true** and **false**.

Identifiers: All identifiers begin with a letter and are followed by any combination of letters and numerals. The maximum length of an identifier is 10 symbols.

Vectors: Vectors can only be subscripted by character or integer types and are referenced using 1 origin.

Built in functions: The built in functions are not yet implemented.

Comments: Comments begin with the characters /* and end with the characters */.

8. Command Request Mode

Purpose: To cause the program, as specified by the switch settings, port name specifications, code names and parameters specifications and the associated XX programs, to be prepared for execution.

Display: The field is not changed, diagnostics and status information are reported in the chalkboard.

Activity: Commands are invoked which cause the source form of the program to be transformed.

Recognized keys:

esc-^a abort, return to UNIX without saving state.
esc-^e exit, return to UNIX and save the current values of the CHiP parameters, switch settings and the port and code names.
esc-^l redraw; the screen is redrawn.
esc-^o output the screen; the BitGraph's raster memory is dumped to a file named BG*xxxxxx* in the current directory, where *xxxxxx* is a random number.
<mode> switch to the indicated mode:

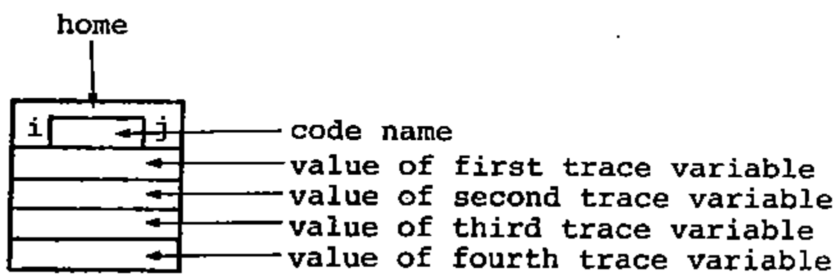
- esc-^c CHiP parameters mode
- esc-^w switch settings mode
- esc-^p port names mode
- esc-^d code names mode
- esc-^v port values mode
- esc-^t trace values mode

<text> is placed on the command line at the position of the cursor.
^h backspace.
^c compile the program whose name is given on the command line; if the command line is blank, compile all programs whose names are mentioned as Code Names for the current phase. The program with name <name> is a file in the current directory with name <name>.x. Errors are reported in a file <name>.2.
^v coordinate the compiled programs whose names are mentioned in Code Names. The assembly code for a program <name> is found in a file in the current directory with name <name>.s.
^a assemble the coordinated programs, one per PE, whose coordinated assembly code is given in files with names of the form PE i, j.s in the current directory. Errors are reported in PE i, j.2.
^t compile the object code for the switch settings for this phase as given by the switch settings specification.
^l load the object code for the PEs and switches into the Pringle emulator.
^g go; begin executing the loaded program; if the command line contains an integer, execute the program for that many steps; otherwise execute it for 10K steps or until it halts.

9. Trace Values

Purpose: To display the current values of the traced variables (peek), to modify those values (poke), and to control the execution.

Display: The code name and the current values assigned to the trace variables of PEs in (a portion of) the lattice for this phase are given in the field. One display format shows boxes representing PEs; the other display format shows boxes representing PEs and lines representing the interconnection structure; a key (^t) toggles between these two. The code name is clipped to five characters (and cannot be changed) and values are shown clipped to the first 10 symbols:



Cursor motions: Gross cursor motions advance the cursor to the home position of the next PE in the indicated direction; fine cursor motions (north and south) move to the first position of the windows for the trace values. 'Home', from a window moves the cursor to the home position of the PE, from the home position in a PE moves to the command line, and from the command line to the home position of Last PE.

Activity: The execution of a loaded program is controlled and the values of the traced variables are displayed. Displayed values can be changed and when execution begins, they will be stored into the memory of the emulator. Execution can be effected in single step units, multiple steps or until a displayed variable changes value.

Limitations: This mode cannot be entered unless a program is loaded.

Recognized keys:

- esc-^a abort, return to UNIX without saving state.
- esc-^e exit, return to UNIX and save the current values of the CHiP parameters, switch settings, and port and code names.
- esc-^l redraw; the screen is redrawn.
- esc-^o output the screen; the BitGraph's raster memory is dumped to a file named BG*xxxxx* in the current directory, where *xxxxx* is a random number.
- <mode> switch to the indicated mode:

esc-^c CHiP parameters mode; causes the current load module to be invalidated
esc-^w switch settings mode
esc-^p port names mode
esc-^d code names mode
esc-^v command request mode
esc-^v port values mode

<text> when entered into any of the trace value windows, becomes the value of the variable when execution resumes; otherwise the text is given on the command line.

^g go; the command line is interpreted as the (integer) number of steps the emulator is to execute; if the command line is blank 10K steps are executed. The new values of the trace variables are displayed at completion of the execution.

^y displays in the auxiliary display area the unclipped value of the window entry.

^t trip; the execution of the emulator resumes until a value of a variable currently being displayed changes.

^b buffers the names of the traced variables and displays them in the chalkboard.

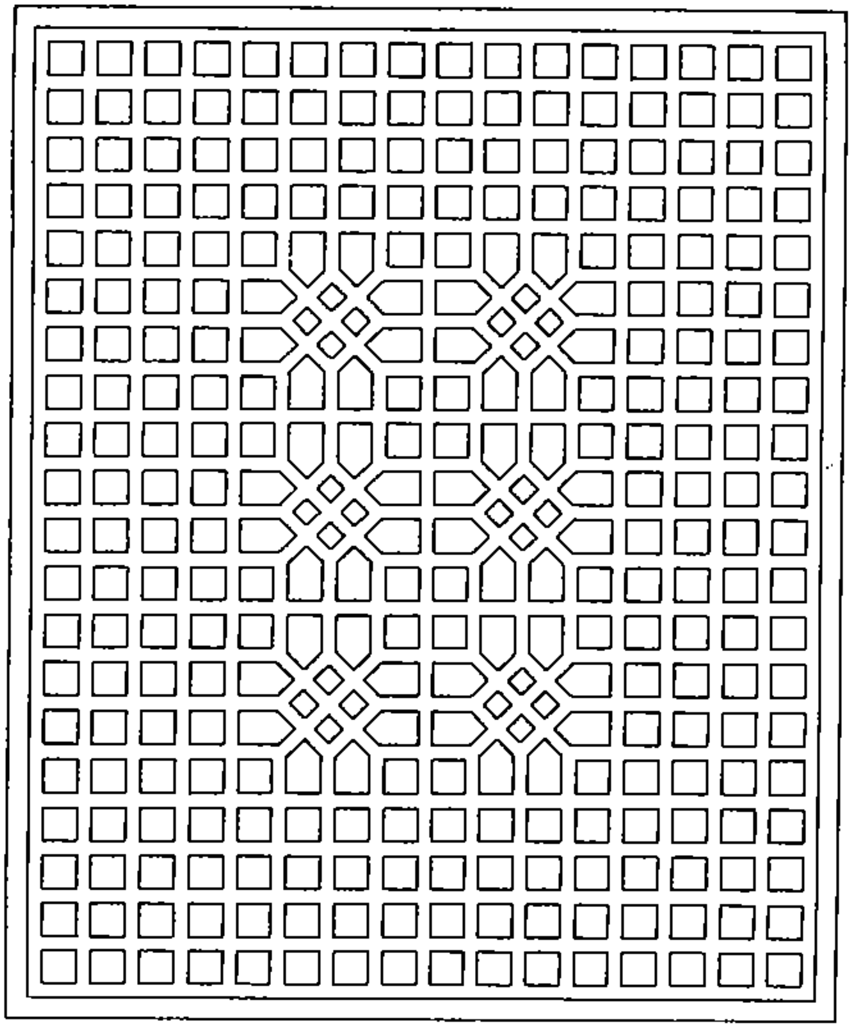
^c center the display so that the PE whose index is given on the command line is as close to the center of the field as possible consistent with the requirement that the field be fully utilized; if the command line is blank, the Last PE is used for centering.

A. Catastrophic Bugs

Like any new, large software system Poker contains many bugs and inconsistant features. Most of these are harmless annoyances that can be easily circumvented. However, a few are serious enough to lead to "mystical" behavior or, worse, to cause "core dumps" that kill the current Poker state. They are documented below.

The cautious user will want, from time to time, to save the current state of an editing mode using esc-^w. If an error causes a core dump, it often happens that the BitGraph will not echo text typed on the UNIX shell. The echo is restored by typing "reset" in the UNIX shell.

1. Switch Settings - cursor motion off screen.
   Cursor motions off the top or right side of the field automatically shift the window. *Cursor motions off the bottom or left side of the field are catastrophic.* Use the center command to manually shift the window.
2. Switch Settings - level anomalies.
   Switches that are set by joining (i.e., two paths that rendezvous at a switch) may not join or may join another path.
3. All modes - esc-^o command.
   The software to dump the screem for the new (3.10) BitGraphs is not yet available and esc-^o is catastrophic for these displays. The copy screen command works only for old (2.0) BitGraphs.



B. Summary of Key Definition

GLOBAL KEYS

esc-^a	abort
esc-^e	exit
esc-^l	redraw screen
esc-^o	copy screen to file
esc-^c	Chip Params mode
esc-^p	Port Names mode
esc-^d	Code Names mode
esc-^v	Port Values mode
esc-^w	Switch Setting mode
esc-^r	Command Request mode
esc-^t	Trace Values mode
<text>	insert text
^h	BACKSPACE

KEYS DIFFERING BY MODE

Switch Setting Mode

^c	center
^d	delete
^e	end
^i	insert
^l	level
^p	phase
^w	write (save)
^x	chase

Port Names & Code Names Modes

^b	buffer
^c	center
^d	delete
^i	insert
^p	phase
^t	toggle (suppress/elicit)
^w	write (save)
^y	display

Command Request Mode

^a	assemble
^c	compile
^l	load
^g	go
^r	coordinate
^t	connect

Port Values & Trace Values Modes

^c	center
^g	go
^r	triggered
^t	toggle (suppress/elicit)
^y	display