CHAMELEON - A CHANGEABLE SMART TERMINAL CASE STUDY

by

Zary Segall and Edward Strasbourger

Technical Report #91
January 1977
ABSTRACT

A smart terminal is proposed and implemented which interactively changes its characteristics to allow a user to efficiently control or evaluate a number of differing applications.

This interactively changeable smart terminal (CHAMELEON) is a microprocessor based system (Motorola M-6800) which interacts with larger computer system resources (Burroughs B-1726).

Software developed for CHAMELEON include a high level interactive Standard Microprocessor Assembler Language (SMAL), an interactive M-6800 simulator, a file-editor, a local microprocessor resident monitor, and a B-1726 resident assisting monitor and data communications system.

CHAMELEON's main contributions are the ability to develop and evaluate a fixed smart terminal application before freezing its smartness, and the great potential for efficient on-line development of software for microprocessor based systems.

Keywords: microcomputer, microprocessor, terminal, interactive, intelligent terminal.
1. THE CHAMELEON CONCEPT

Methods for distributing data processing power have evolved toward placing the processing power closer to the ultimate user.

Steps in this evolution are:

a. Totally centralized processing center.

b. Processing center with directly connected remote peripherals.

c. Processing center with remote peripherals via front-end processor or channel.

b. Processing center with remote editing and formatting terminals - "smart terminals".

The paper will discuss the 4-th step (smart terminals), the most advanced step toward distributed computing, and will evolve a further step - The Changeable Smart Terminal.

Smart terminals accomplish the preliminary level processing apart from the central processor, affording two main advantages:

* Less loading of the central processor promoting higher throughput and better response times.

* Greater terminal flexibility promoting better human interaction.

The main role given to the smart terminal is to provide facilities aiding program preparation (editing), program debugging, interactive execution, and local actions.

Terminals may be either on-line with a central data base or off-line with only a local portable data base. In other words an on-line smart terminal communicates directly with a central processing facility for exchange of data, while an off-line smart terminal creates and/or interrogates files on local non-volatile secondary storage such as magnetic tapes which are transported between terminal and central processor.

There is an expanding variety of smart terminal application such as business transaction, data entry, bank tellers or air line reservation.
Implementation of smart terminals utilize two techniques:

* Random logic (hardware logic).
* Programmable logic (generally micro-processor based logic).

Micro-processor based logic appears to be an attractive solution, as compared with random logic, in a smart terminal implementation because of:

* Flexibility stemming from reprogrammability.
* Slow speed requirements resulting from slow man-machine interaction.
* Short development time since most of the development entails mainly software.
* Low cost, small size and low power requirements.

Random logic based smart terminals are generally rigid systems designed for a unique application. However, microprocessor based smart terminals may be reprogrammed allowing changes in its task. This idea leads naturally to the concept of the Changeable Smart Terminal, i.e. an on-line smart terminal with a task as defined interactively by a user loading and executively an application program. This application program is a software system interacting with a resident operating system to provide the specific smartness or functions desired (Fig. 1).

The resident operating system is a collection of resident programs which provide:

* Loading and executing application programs.
* Preparing application programs.
* I/O services.
* Memory allocation and protection.
* File management.

This concept requires a non-dedicated structure to allow for easy implementation of any smart terminal application. A block diagram for a Changeable Smart Terminal is shown in Figure 2.

The design philosophy behind each component is as follows:

The communication line is the physical connection between the Changeable Smart Terminal and Central Processor. Line characteristics
Figure 1. The Changeable Smart Terminal - Software Concept

Figure 2. The Changeable Smart Terminal - Hardware Concept
should be designed to satisfy two requirements:

* Compatibility with most computer systems.
* Ability to transmit files within a reasonable period.

Most available communication lines have the following features:

* Transmission type - serial asynchronous and/or synchronous.
* Transmission speed - 110 to 9600 baud; minimum speed of 1200 baud is necessary for file transmission.
* Line discipline - multidrop or multipoint.
* I/O protocol - polling or contention.

The Changeable Smart Terminal could be adopted to the majority of combinations of the above features by means of software modifications.

The computer link functions are:

- transferring information between the communication line and the processing unit.
- transforming parallel information received from the Processing Unit to serial information required by the communication line and vice-versa.
- providing the information status and error checking.

The Processing Unit is a micro-processor which executes system and application programs stored in Local Memory.

The Local Memory can be divided into the following levels:

- ROM - containing operating system components such as:
  * Bootstrap;
  * self tester;
  * I/O software interface;
  * monitor.

  If the application program becomes fixed, then it should also be stored in ROM.

- RAM - containing the application programs and local data base.

  In the development phase of the Changeable Smart Terminal the operating system should reside in RAM providing for easy changing.

Secondary Local Memory could be of the tape cassette and/or floppy disk
type and can be shared between parts of the operating system and application programs. However, the main role of the Secondary Local Memory is mass-storage of a local data base.

If Secondary Local Memory is not available then the Central Processor Secondary Memory may be used. In this case the simplicity and economy will be offset by the extra loading of the Central Processor and by the limited complexity of application programs which could be locally executed.

- Terminal Link - This is the connection between the Processing Unit and the Terminal. The role and the structure of the Terminal Link is closely related with the Terminal type. The Terminal type is determined by the type of application that the system must perform.

The Terminal must have the following two features:
* to allow the preparation of application programs;
* to be suitable for execution of a specific type of application program.

The operating system has two distinct parts:
- Local Operating System - at the Changeable Terminal Level;
- Assisting Operating System - at the Central Processor Level.

The operating system tasks are:
- I/O protocol handling between Changeable Smart Terminal and Central Processor;
- I/O handling between Processing Unit and Terminal;
- memory protection and allocation;
- file management;
- program developing aids.

The assisting systems purposes are:
* provision of utilities for the user;
* provision of standard I/O routines transparent of particular I/O device requirement details;
* assisting the execution of application programs;
* allowing interaction with the Central Processor files.
These hardware and software concepts along with the equipment available have dominated the design decisions in the development of CHAMELEON - A Changeable Smart Terminal.

2. CHAMELEON - IMPLEMENTATION

Chameleon is a changeable smart terminal which may function as any specific smart terminal under user program control. It is implemented as a microprocessor-based system using the M-6800 family of microprocessors.

Chameleon is a smart terminal where functions are defined by interactively loading and executing a certain application program. The system can be seen also as a framework for evaluating smart terminal systems, based on the use of microprocessors. By loading and executing an application program, Chameleon may become a specific smart terminal. In this way Chameleon provides the possibility of studying and evaluating the efficiency of the solution for a specific application-oriented smart terminal.

For example, Chameleon may become a Secretary Helper terminal by loading and executing locally or application program with the following components:

* Letter writing and correcting;
* Forms completion;
* Calendar service;
* Diary service;
* Time schedule for the manager;
* Filing service. (On-line using the main computer data base.)

Chameleon has been also designed to allow interactive developing of application programs (editing and debugging).

One of the most interesting applications of Chameleon is the development of other microprocessor-based systems.

Chameleon has been designed and implemented in the Technion's Computer Science Department and uses the available equipment in the Department.
Chameleon is connected to the Department's B-1726 Computer by means of its Single-Line Control. Any of the available RS-232 compatible terminals (TTY, INFOTON, TEKTRONIX 4013, IBM 2741) can be connected to the Chameleon, as shown in Figure 3.

![Diagram showing Chameleon environment](image)

Figure 3: CHAMELEON environment - block diagram

The Chameleon system works in two modes:

* remote and
* local.

In the remote mode Chameleon uses the secondary storage of the B-1726 (discs) for storing data. Also, Chameleon can load and execute programs located on the B-1726 disc files. Applications' programs loaded from the B-1726 and executed on Chameleon system determine the terminal's functions.

One particular mode is the teleprocessing mode where the Changeable Smart Terminal becomes a B-1726 standard terminal addressable by the B-1726's operating system.

If the application program at execution time does not need a large data base, the Chameleon can execute the program locally liberating the B-1726 resources. All the application programs run in Chameleon under control of
the local operating system which handles:

* I/O protocol with the B-1726;
* memory allocation and protection;
* file management.

Also, there is on the B-1726, an assisting software system providing:

* I/O protocol handling;
* interface between assisting programs and B-1726 operating system;
* application assisting programs.

The architecture of the Chameleon is presented in Figure 4.

Figure 4. - CHAMELEON Architecture
THE CHAMELEON SOFTWARE CONCEPT

The Chameleon software system has been designed to satisfy two main goals:

* Providing a user transparent I/O software interface between Chameleon and B-1726 and also between Chameleon Processing Unit and the associated Terminal (the I/O software interface).

* Providing a software environment for development and execution of application programs (the Monitor).

THE I/O SOFTWARE INTERFACE

The I/O software interface has two functional parts:

* The Chameleon I/O software interface;
* The B-1726 I/O software interface.

The Chameleon I/O software interface is composed of a set of system subroutines residents in Chameleon Local Memory which alleviates the user of knowing about the sophisticated aspects of I/O protocol between Chameleon and the B-1726, and also provides standard and efficient procedures accomplishing the I/O between the Chameleon Processing Unit and the associated Terminal.

The B-1726 I/O software interface contains programs which run in the regular environment of the B-1726 Operating System (Master Control Program - MCP) using its facilities for all I/O handling. The programs which are basically responsible for the Communication System in B-1726 are:

* Network Controller - NC;
* Message Controller System - MCS.

NC interfaces with the Single Line Control hardware and is responsible for I/O protocol handling.

MCS supervises the flow of messages between Chameleon and the communication related tasks running in the B-1726.

NC is written in HDL (Network Development Language) which is a high level communication programming language and MCS is written in SDL (System Development Language), a PL/1 like language.
For overall understanding of the software I/O interface concept let us have an examplification.

Assume that one user has the problem of transmitting a text from Chameleon (terminal) to B-1726. For accomplishing this task two programs are necessary: one running in Chameleon and the other in the B-1726.

a. Program in Chameleon

* Using the GET.FROM.TERMINAL subroutine called in the following standard way:

```
CALL GET.FROM.TERMINAL
DATA DOUBLEWORD BUFFER ADDRESS
DATA WORD MAXIMUM BUFFER LENGTH
```

Text typed at the Chameleon Terminal is stored beginning from BUFFER ADDRESS in Local Memory.

* Using the PUT.TO.B1726 subroutine called in the following standard way:

```
CALL PUT.TO.B1726
DATA DOUBLEWORD BUFFER ADDRESS
```

The contents of the Buffer is transmitted to B-1726 taking care of I/O protocol error handling.

b. Program in B-1726

* Using the READ and FILE instructions:

```
FILE CHAMELEON (DEVICE=REMOTE)
READ CHAMELEON
```

an SDL program reads one message from the file associated in MCS with Chameleon.

* The READ command is transmitted by MCS to NC.

* The NC reads the message from the Single Line Control according to the 1/O protocol and puts it in the NC associated file.

* The MCS takes the message from the NC associated file and transfers into the user program file (named in our example CHAMELEON).

As it stands from this example, in order to perform the proposed task the user has only to use the standard I/O routines in CHAMELEON (GET.FROM.TERMINAL, PUT.TO.B1726) and to write a short B-1726, SDL program which
utilises the services of NC and MCS.

By means of this software I/O interface concept, a standard, economic and efficient tool is provided to the Chameleon user absolving him of being aware of the I/O problems.

The above example describes the simplest interaction between one Chameleon system and one application program. However, if more than one Chameleon system or more than one application program must be accessed additional information (such as terminal number) should be provided to the MCS.

THE MONITOR

The Monitor programs allow the user to develop interactively (assembling, compiling, loading, editing, debugging, executing) the application programs along with file management.

Considering the physical location of the Monitor programs we can divide the Monitor in two parts:

* The Chameleon Monitor resident in Local Memory;
* MCS located in B-1726 Computer.

The Chameleon Monitor provides means of determining one of the following three modes of operation:

* Command mode;
* Teleprocessing mode;
* Local Processing mode.

In the Command mode the Chameleon user may specify a number of commands which determine either the execution of one of the Chameleon Monitor functions (loading file from B-1726, dump file to B-1726, etc.) or the transition to one of the two other modes.

In the Teleprocessing mode Chameleon becomes a B-1726 console accessible by B-1726 operating system (MCP).

The Local Processing mode designates the mode in which Chameleon executes locally an application program.
The message Control System (MCS) allows user interaction and control with MCP and other programs located in B1726 Computer and running in the MCP environment. These programs can be either production programs (assisting application programs) or programs helping the user in developing new application programs. Some of the helping programs are:

* SMAL
* CROSS ASSEMBLER
* SIMULATOR
* FILE EDITOR.

THE SMAL CONCEPT

Assembly languages were developed with greatly enhanced the ability to read, write and debug machine language programs. Then compiler language appeared further enhancing these programming characteristics and at the same time hiding the computer structure from the programmer.

Today, most programs are written in compiler level language except when it is desired to explicitly exploit the computer structure, such as in systems programming and in programming severely memory limited computers such as most microcomputer systems and many mini and small scale computer systems. Recently, higher level assembly languages have appeared for system programming such as PL/360 [7] and AL [6]. Another interesting development is the formulation of a higher level assembly language MIL [3] as the assembly language for the B1700 computer. An advantage of MIL is the fact that it is completely word and phrase oriented rather than partially mnemonic abbreviation oriented as are the other assembler languages.

Compiler level languages such as FORTRAN and COBOL are reasonably standardized and algorithms written in these languages can be understood by someone with little knowledge of compilers or computer structure. Users of assembly language, on the other hand, must have some knowledge of computer structure.

There is however, a large need for standardization of assembly language to facilitate
* appraising different computers;
* transporting software among different computers;
* investigating improved computer structure.

SMAL is designed to be a Standard high level Assembler Language, being oriented toward micro and minicomputers.

Currently we have an operational SMAL assembler for Motorola M6800 microprocessor on B1726 computer and are in process implementing SMAL assemblers for Intel 8080, PDP8, PDP11, Z80 and T19900.

The operational SMAL is one pass and can be used interactively with line by line error detection and correction in addition to normal batch usage.

LANGUAGE DEFINITION

A lower case word is a nonterminal symbol, i.e. is explained on this page. Upper case letters and special characters are terminal symbols.

Brackets [ ] denote a choice of one of the included options, unless there is only one option in which case the brackets mean to either include or to omit the option as desired.

Triple dots ... means an indefinite repetition of the preceding option.

A "comment" is any characters in a line following a semicolon.

A "register" is one of a subset of the available registers. For example in the M6800 a "register" is one of a subset of \{(A,B,X,PC,SP,CC)\}, often the subset \{(A,B)\}, optionally preceded by the word REGISTER.

A "flag" is any of the flag and status bits which vary from computer to computer such as CARRY, OVERFLOW, or LINK.

A "number" may be decimal such as 25, or hexadecimal such as HEX 25 = 37, or octal such as OCTAL 25 = 21, or binary such as BINARY 1101 = 25.

A "symbol" is a string of continuous alphanumeric characters which must begin with an alphabetic and may include periods as concatenators such as GET.NEXT.SYMBOL.

A "label" is a symbol immediately followed by a colon such as NEXT.WORD.
A "memory" is a location expressed as a symbol such as M, or a number such as HEX 25, or a displacement from an index value such as (X) - 3, or an indirect reference such as (M) or ((X)+5).

A "character" is a single ASCII or EBCDIC character such as "T" or "I". Default is ASCII unless overridden by an EBCDIC instruction.

A "string" is a string of characters surrounded by quotes such as "SMAL ACTIVE".

A "line" of SMAL code is [label] [instruction] [comment] which may be blank or may contain just a label or just an instruction or just a comment or any two parts or all three parts. Only the IF instruction may be split into two lines with the IF part appearing on one line and the THEN part appearing on the next line.

A "source" is

\[
\begin{align*}
\text{register} & \quad \text{character} \\
\text{NUMBER} & \quad \text{ADDRESS OF} \\
\text{CONTENTS OF} & \quad \text{memory}
\end{align*}
\]

where a number is a value unless preceded by CONTENTS OF and a symbol is a location unless preceded by NUMBER or ADDRESS OF

A "condition" is

\[
\begin{align*}
\text{flag} & \quad \text{[BIT] [IS] } \text{SET} \\
\text{ NOT} & \quad \text{flag} \\
\text{RESULT} & \quad \text{test} \\
\text{source} & \quad \text{WHERE} [\text{INCREMENTED/DECREMENTED} \text{By number}] \quad \text{IS} \quad \text{test} \quad \text{comparison} \\
\text{MATCHES} & \quad [\text{ANYWHERE/NO/HERE}] \quad \text{source}_2
\end{align*}
\]

where "test" is [NOT] ZERO and "comparison" is [NOT] [> <] source

"flag" is

\[
\begin{align*}
\text{CARRY} & \quad \text{INTERRUPT} \\
\text{NEGATIVE} & \quad \text{in the } \text{H6800} \\
\text{ZERO} & \quad \text{HIGH/LOW}
\end{align*}
\]
SMALL instructions

ADD [DECIMAL]: source TO register [WITH CARRY]
AND source TO register
ASCII [MEMONICS]
CALL symbol
CLEAR [flag or register or memory]
COMPLEMENT flag or register or memory
DATA [DOUBLEWORD] [symbol] [symbol] ...
DECIMAL ADJUST register
DECREMENT [register or memory] By number
DEFINE symbol As source
DISABLE INTERRUPT
EBCDIC
ENABLE INTERRUPT
EXCHANGE [register or memory] WITH [register or memory]
FILL number [WORDS]
GO TO symbol
HALT
IF condition THEN any instruction other than IF ELSE any inst
INCREMENT [register or memory] [BY number]
INPUT [FROM] source TO [register or memory]
INTERRUPT
LOCATION number
MOVE source TO [register or memory]
NEGATE [register or memory]
OR source TO register
OUTPUT [FROM] [register or memory] TO [number or memory]
POP [memory] INTO register
PUSH register [ONTO MEMORY]
REGARDLESS IF condition
RESERVE number [WORDS]
RETURN [FROM [INTERRUPT or SUBROUTINE]]
ROTATE [register or memory] [left or right] [number BITS] [BYPASSING CARRY]
SET flag
SHIFT [register] [left] [ARITHMETICALLY] [number BITS] [BYPASSING CARRY]
source FROM register [WITH BORROW]
SWAP [register or memory] WITH [register or memory]
WAIT FOR INTERRUPT
XOR source TO register
SAMPLE PROGRAM

Program BUBBLE.SORT to be called with array to be sorted starting in location ARRAY and with address of last element in doubleword location END.OF.ARRAY and END.OF.ARRAY + 1.

MNEmonic Program

<table>
<thead>
<tr>
<th>BUBBLE.SORT</th>
<th>BUBBLE.SORT</th>
<th>Structured SMAL Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT.PASS</td>
<td>NEXT.PASS</td>
<td>PROCEDURE BUBBLE.SORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX #ARRAY</td>
<td>MOVE ADDRESS OF ARRAY TO X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE (X) TO A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLEAR SWITCHED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE (X) TO B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF A IS NOT &gt; B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE A TO (X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE A TO B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF X IS NOT-END.OF.ARRAY</td>
<td>END SWITCH</td>
</tr>
<tr>
<td></td>
<td>THEN GO TO NEXT.PAIR</td>
<td>MOVE B TO A</td>
</tr>
<tr>
<td></td>
<td>IF SWITCHED IS NONZERO</td>
<td>THEN undo NEXT.PAIR</td>
</tr>
<tr>
<td></td>
<td>THEN GO TO NEXT.PASS</td>
<td>RETURN FROM BUBBLE.SORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA B X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC SWITCHED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA A X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPX END.OF.ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE NEXT.PAIR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TST SWITCHED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE NEXT.PASS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCHED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END.OF.ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SMAL Program:

<table>
<thead>
<tr>
<th>BUBBLE.SORT</th>
<th>NEXT.PASS:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOVE ADDRESS OF ARRAY TO X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE (X) TO A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLEAR SWITCHED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE (X) TO B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF A IS NOT &gt; B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE A TO (X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOVE A TO B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF X IS NOT-END.OF.ARRAY</td>
<td>END SWITCH</td>
</tr>
<tr>
<td></td>
<td>THEN GO TO NEXT.PAIR</td>
<td>MOVE B TO A</td>
</tr>
<tr>
<td></td>
<td>IF SWITCHED IS NONZERO</td>
<td>THEN undo NEXT.PAIR</td>
</tr>
<tr>
<td></td>
<td>THEN GO TO NEXT.PASS</td>
<td>RETURN FROM BUBBLE.SORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA B X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC SWITCHED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA A X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPX END.OF.ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE NEXT.PAIR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TST SWITCHED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE NEXT.PASS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCHED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END.OF.ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Structured SMAL Program:

<table>
<thead>
<tr>
<th>BUBBLE.SORT:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT.PASS:</td>
<td>MOVE ADDRESS OF ARRAY TO X</td>
</tr>
<tr>
<td></td>
<td>MOVE (X) TO A</td>
</tr>
<tr>
<td></td>
<td>CLEAR SWITCHED</td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
</tr>
<tr>
<td></td>
<td>MOVE (X) TO B</td>
</tr>
<tr>
<td></td>
<td>IF A IS NOT &gt; B</td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
</tr>
<tr>
<td></td>
<td>Increment X</td>
</tr>
<tr>
<td></td>
<td>MOVE A TO (X)</td>
</tr>
<tr>
<td></td>
<td>MOVE A TO B</td>
</tr>
<tr>
<td></td>
<td>IF X IS NOT-END.OF.ARRAY</td>
</tr>
<tr>
<td></td>
<td>THEN GO TO NEXT.PAIR</td>
</tr>
<tr>
<td></td>
<td>IF SWITCHED IS NONZERO</td>
</tr>
<tr>
<td></td>
<td>THEN GO TO NEXT.PASS</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>STA B X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>INC SWITCHED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>STA A X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TAB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TBA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>CPX END.OF.ARRAY</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE NEXT.PAIR</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TST SWITCHED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE NEXT.PASS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCHED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>END.OF.ARRAY</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ARRAY</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Technion - Computer Science Department - Technical Report CS0091 - 1977
THE CROSSASSEMBLER.

A Crossassembler for Motorola M6800 has been implemented on B1726. The assembler language definition and convention are the same as presented in "M6800 Microprocessor Programming Manual" [12] except expressions which are not yet implemented. The Crossassembler is written in SDL and the results of assembling are the listing on the line printer along with the assembled program in the file M6800CODE located on B1726 system disk.

SIMULATOR

The SIMULATOR is an interactive program designed to simulate on B1726 the operation of the M6800 microprocessor. This program works on a copy of the object code file resultant from cross assembler or SMAL.

When the user asks for execution of the SIMULATOR program the SIMULATOR's Program Counter is set to the starting address of the program under simulation, (the first record from the object code file contains the program starting address), while the other registers are set to zero.

The SIMULATOR program once started will stop either on Break-point or as result of simulated execution of Wait-for-interrupt instruction.

The maximum memory area which the M6800 object code may occupy and use is 65k/bytes.

The SIMULATOR program can be operated interactively using a number of commands. The commands which can be used with SIMULATOR program follows:

1. CR
   * Function:
     - change registers.

2. CC
   * Function:
     - change condition code.

3. BP
   * Function:
     - break points.

4. CM
   * Function:
     - change memory.

5. ST
   * Function:
     - start the simulation from the indicated location.
6. **DR**
   
   * **Function:**
   - display register.

7. **TR**
   
   * **Function:**
   - trace the registers by printing on B1726 line-printer the contents of registers after each instruction.

8. **DM**
   
   * **Function:**
   - display memory location.

9. **ST**
   
   * **Function:**
   - single instruction

10. **EP**
   
   * **Function:**
   - end the SIMULATOR program.

Special emphasis has been put in the modularity of SIMULATOR program permitting changes and addition to the existing commands.

A natural addition is the simulation of the other modules of M6800 families such as PIA and ACIA modules. It will also be useful to implement time measures on the simulating process.

**FILE-EDITOR**

The File-Editor (EDIT) is an interactive Application Program designed for card-image file processing. It allows users at terminals to create and/or alter B1726 disk files containing card-image records. The resulting files may be used as source input to the various compilers in the B1726 or as input Data to any program running on it. Note that these files are numbered (columns 73 through 80 contain the sequence number) in ascending order, a format conforming to that of card-input to compilers.

Two modes of operation exist under EDIT, Input mode and Command mode.

In input mode the user enters line after line of input to his file, all text being treated equally; command mode allows the user to make modifications (editing) in the form of text changing, line insertion and line deletion, with the additional capability of checking (listing) parts or all of the edited file.
When the user enters EDIT in order to create a new file, the active mode is input. Otherwise the starting mode is Command mode. When changing modes, as well as at the beginning, a "mode message" tells the user which mode is currently active ("*n* for Input, "*C* for Command mode). A null line is the switch between the two modes. The commands that can be used when in EDIT-Command mode follow:

1. **BOTTOM (alias B)**
   Function: Move the CLP to the bottom (last line) of the file.

2. **CHANGE (alias C)**
   Function: Replace string of characters by other strings of characters. The user is notified if the string to be replaced is not found.

3. **DELETE (alias DEL)**
   Function: Delete a line or range of lines. The user is notified if the line(s) do not exist.

4. **DOWN (alias D)**
   Function: Move the CLP down in the file. The user is told if by this operation the bottom (last line) of the file is reached.

5. **END (alias E)**
   Function: Exit the File-Editor (EDIT), return terminal responsibility to the MCS (Terminal Command mode). If changes have been made to the file and it hasn't been "SAVED" (see SAVE below), the user is warned and asked to confirm the END command.

6. **FIND (alias F)**
   Function: Locate the first occurrence of a character string, starting at the line pointed at by the CLP and up to the last line of the file. The user is told if the requested string is not found in this range of lines.

7. **INPUT (alias I)**
   Function: Enter Input mode for appending or inserting input lines. The user is told if lines cannot be inserted.

8. **LIST (alias L)**
   Function: Display line(s) at the terminal. The user is told if the line(s) requested do not exist.
9. **RESEQ (alias RES)**
   Function: Renumbers the whole file with new sequence numbers.

10. **SAVE (alias S)**
    Function: Create a permanent copy of the Edit-file on the B1726's disk. This command must be entered in order to have a new copy of a file, containing all the changes made during the editing session.

11. **TOP (alias T)**
    Function: Move the CLP to the top (first line) of the file.

12. **UP (alias U)**
    Function: Move the CLP up in the file. The user is told if by this operation, the top (first line) of the file is reached.

The File-Editor has been developed, along the guidelines described above. Special emphasis is put in its modularity so as to facilitate changes and additions to the existing commands.

---

**THE CHAMELEON LOCAL MONITOR**

The Chameleon Local Monitor program, located in the Chameleon Local Memory allows the user to choose interactively the work mode, and assists the loading and execution of application programs.

User may specify one of the following commands:

- the "+" key: Invoke Chameleon Local Monitor (enter command mode);
- the "EOT" key: Invoke Debug Monitor;
- **TP** Teleprocessing mode;
- **LD [file-name]** Load application program from file-name located on B1726 disk;
- **GO [location]** Execute local program starting from [location].
- **EX [file-name]** Combined action of LD and GO.

"+" and "EOT" commands are always active while the other commands are active only in Command Mode. "+" will activate the Chameleon Local Monitor under any condition going into Command mode. "+" activates the initialization routine of the Chameleon Local Monitor, which initializes all hardware interfaces, and registers bringing the system to a well defined initial state without affecting Local Memory contents.
THE MESSAGE CONTROL SYSTEM

The Message Control System is a program located on B1726 which is designed to supervise messages from a number of smart terminals such as Chameleon.

This program monitors input from all Chameleons defined in the system, interprets the commands entered at them, checks these commands for validity through interaction with MCP, and either executes the command or notifies the user of the errors found in it.

The MCS commands are interactively accessible for the Chameleon user in the Teleprocessing mode, but all the on-line Chameleon functions necessitate the MCS presence.

The MCS commands can determine either the execution of a MCP command or the execution of an application program. Execution of an application program as result of the EXECUTE command causes transfer of responsibility for the Chameleon from the MCS to application program. When the application program ends its execution, it has to notify the MCS so responsibility for input from Chameleon is returned to the MCS.

The MCS Commands

The following commands were designated in order to simplify B1726 access to the Chameleon user. The command language is interpreted by the MCS and is easily expandable.

1. **BYE** (alias **B**)
   
   Function: Logically disconnect Chameleon from system

2. **Compile** (alias **C**)

   Function: Compile a source file, and/or save it, and/or execute it.

   Example: Co U/SDL/TEST:0 = U/SDL/LIST SDL LI will "compile to library" the SDL source to disk file "U/SDL/TEST and write listing output on disk file "U/SDL/LIST".
3. **DELETE (alias DEL)**

Function: Deletes a disk file from a disk directory. If the file does not exist or cannot be deleted an error message is sent to Chameleon.

Example: `DEL U/PROGRAM/A`
will remove the disk file `U/PROGRAM/A` from the disk's directory.

4. **EDIT (alias E)**

Function: Invokes the File-Editor to create or make changes to a card-image disk file.

Example: `EDIT SOURCE/PROGRAM`
will invoke the file editor for disk file "SOURCE/PROGRAM" assumed to be already in disk's directory.

5. **EXECUTE (alias EX)**

Function: Invokes an interactive program for execution.

Example: `EX U/INTERACTIVE/PROGRAM CARDS DISK`
will invoke the interactive program in disk file "U/INTERACTIVE/PROGRAM" for execution, also specifying that the file with internal name CARDS be assigned to the System's disk.

6. **RENAME (alias REN)**

Function: Renaming of disk files. Existence of files is checked and the user gets the appropriate messages according to the files' status.

Example: `REN PROGRAM/A PROGRAM/B`
disk file name `PROGRAM/A` will be changed to `PROGRAM/B`.

7. **SMAL**

Function: Invokes the SMALR (remote SMAL) high level cross assembler for interactive work.

The input is entered line by line from Chameleon and SMAL syntax errors (if any) are sent to Chameleon. At the end of
source program (indicated by user typing "?END") the object code will be in the disk file named M6800CODE and an assembly listing will be printed on the Line-Printer.

Example: SMAL WORK/FILE.
SMALR has been invoked allowing the user to introduce interactively the source SMAL program which will be stored in the disk file name "WORK/FILE".

8. SIMULATE (alias SIM)

Function: invoke the M6800 Interactive Simulator

Example: SIM
will invoke the M6800 interactive Simulator.

9. XFER (alias X)

Function: transfer (copy) files between different storage media in the form of card images.

Example: - XFER FILEA FILEB disk file FILEA will be copied to disk file FILEB
- X FILEA
disk file FILEA will be printed on the Line-Printer.
- X *
will reproduce (echo) each input line from the Chameleon to Chameleon itself.

10. ZIP (alias Z)

Function: pass a command to the MCP, no checks are made on the text passed in this command.

Example: ZIP 1DS 2DS
will "DS" the programs with mix numbers 1 and 2.
CONCLUSION

The system is currently operational and is useful as a general purpose changeable smart terminal for B-1726.

In developing a specific application (a particular smart terminal), the following steps may be interactively performed:

(a) Using SMAL (High Level Assembler) in the interactive way a syntactically debugged application program can be created in the B-1726 disk.

(b) Using the interactive SIMULATOR a first run test and debug can be performed.

(c) Errors founded in the simulation process can be corrected using the interactive FILE-EDITOR program.

(d) The above sequence of steps may be repeated till no more errors are encountered.

(e) Using the CHAMELEON Local Monitor interactively the object file is loaded into CHAMELEON's Local Memory.

(f) Using the CHAMELEON Debug Monitor the application program is debugged and executed locally resulting in increasing the terminal smartness or changing the terminal application potential.

Further enhancements to CHAMELEON that will augment its power would be the incorporation of local mass storage and creating of a cross compiler.

Two main contributions result from the development and use of the changeable smart terminal system concept:

- Ability to develop and evaluate any particular smart terminal application before freezing the smartness.

- Great potential for efficient on-line development of software for microprocessor based systems.
REFERENCES


