Document Nbr: CSI426/kayproII-RD2.06
Date: December 5, 1998
Copy Nbr: ____

DESIGN DOCUMENT

FOR THE

CSI426/KAYPRO II EMULATOR

PROPRIETARY & CONFIDENTIAL

Distribution List

Name	**Organization**	**Signature**
Dr. Charles Howerton	**Metropolitan State College**	_____

Table of Contents

Tables and Figures

Figure 1 Kaypro II Functional View..... 6
Figure 2 Kaypro II Emulator Java Applet Transfer..... 8
Figure 3, User Interface Example..... 9
Figure 4 Output Screen Example 19
Figure 5, CPU and Opcodes..... 21
Figure 6, Opcode Object..... 27
Figure 7, Port Emulation..... 29
Figure 8 Kaypro II Bank Switching 30
Figure 9, SIO/PIO Model..... 34
Figure 10, Hardware Port Connectivity..... 40

1. Introduction

This section contains the overview, system identification, and scope of the Design Document.

1.1 Overview

This document describes the software design for the Kaypro II emulator.

This document presents an overview of the Kaypro II emulator design. The emulator is a Java based implementation of a Kaypro II computer system. Java is a language that allows remote programs to be executed on a client computer via the World Wide Web. The Kaypro II emulator resides on a remote host machine. The user may run the emulation on a compatible browser from their own computer system.

The implementation features a number of convenient and useful features, including:

- Printer port simulation
- Debugging:
  - Hardware style breakpoints
  - Opcode level debugging
  - CPU register display
- Memory dump utilities
- Dual virtual diskette drives with CP/M pre-loaded
- Real and fast mode video options

In addition to the above, the system shall be coded in such a way that it is expandable. This expandability requires changes to the source code. The modular nature of the implementation allows easy modification.

1.2 Scope

The Kaypro II emulator utility shall be a new development effort.

2. System Requirements

2.1 System Overview

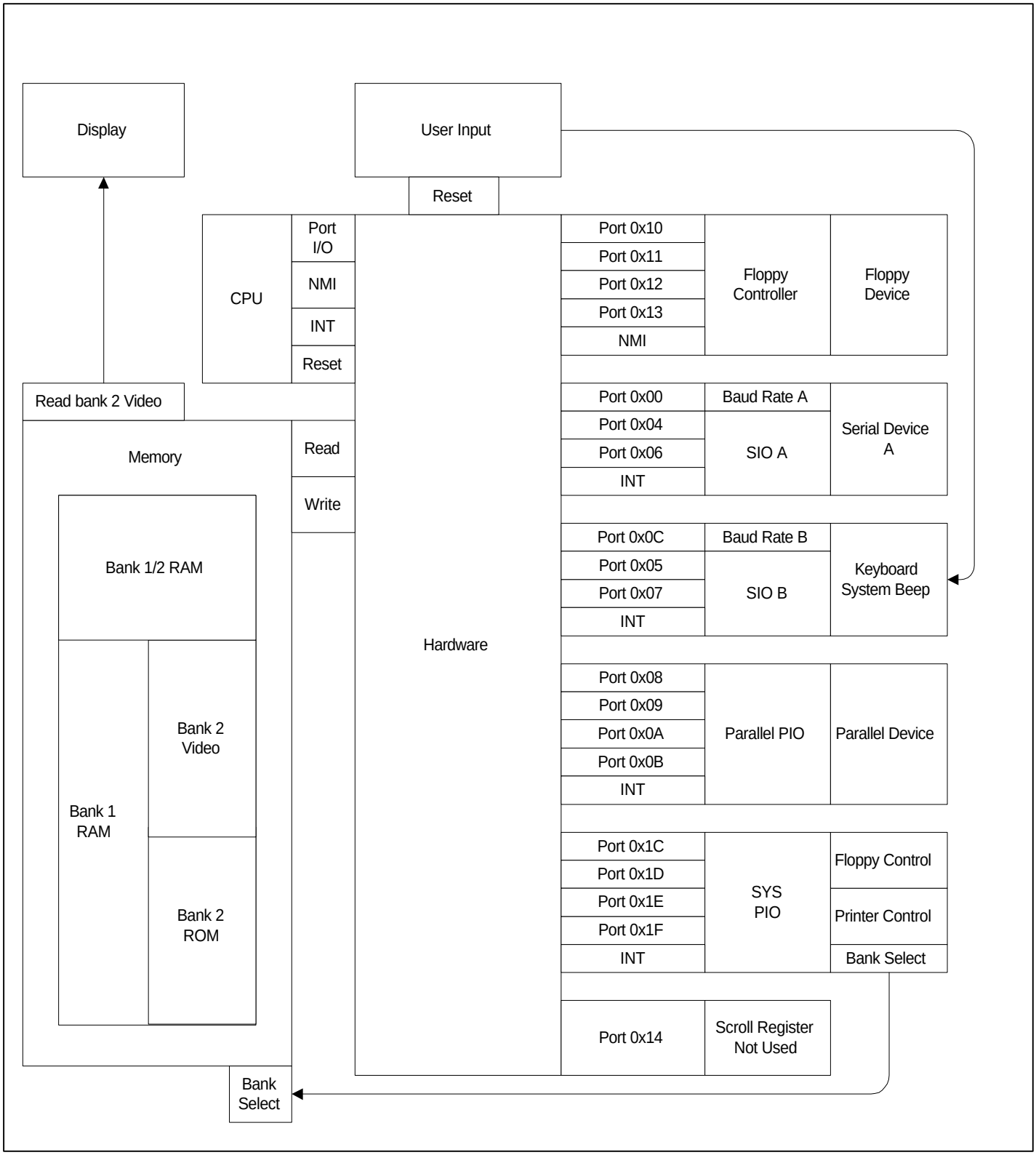


Figure 1 Kaypro II Functional View

2.1.1 Concept of Operations

The Kaypro II is a Z-80 based computer system. It contains dual floppy drives, serial I/O, a monochrome screen, keyboard, a printer port, memory and control logic.

The Z-80 processor executes pre-programmed instructions read from memory. Serial I/O provides an interface to serial devices. Serial I/O also provides interface to the built-in keyboard and system speaker.

The Kaypro II contains a monochrome screen. The screen has no real graphic capability, but can display inverse characters.

The Kaypro II includes two built-in floppy disk drives.

The printer port enables communication with a Centronics compatible printer. A secondary printer port acts as an interface for controlling internal functions such as:

- Floppy disk selection
- Floppy drive motor control
- RAM/ROM/video RAM bank selection
- Printer control signals

The Kaypro II supported 64K of RAM. In addition, the system also included a system ROM and memory mapped video.

The Kaypro II control logic included a dual baud-rate generator for serial data rate adjustments, a character generator ROM, and other discrete control logic.

2.1.2 Base Objects

The Kaypro II emulator will consist of 4 main objects

- The User Interface (UI). This is the user entry system. It allows the user to manipulate the emulation.
- The CPU. This is an emulation of the Z-80 microprocessor
- The SIO. This is the serial I/O
- The PIO. The parallel I/0
- The hardware. This object contains the main memory. It also acts as a communicator between the other objects.

These objects will be describe in greater detail later.

2.2 Functional Requirements

2.2.1 Function

- The emulation program shall emulate the Kaypro II model of the Kaypro product line
- The user shall manipulate the emulated version just as they would the original
- The emulation shall contain debugging features, in addition to the original functionality

2.2.2 Expandability

The system shall be expandable. That is, its design shall be easily upgraded. This allows improvements, variations and upgrades to be easily coded and implemented.

- The Kaypro II emulation shall be coded in such a way as to facilitate easy additions and expansions to the system.
- The emulation shall be coded in a modular way; such as logical Java classes

2.2.3 Platform

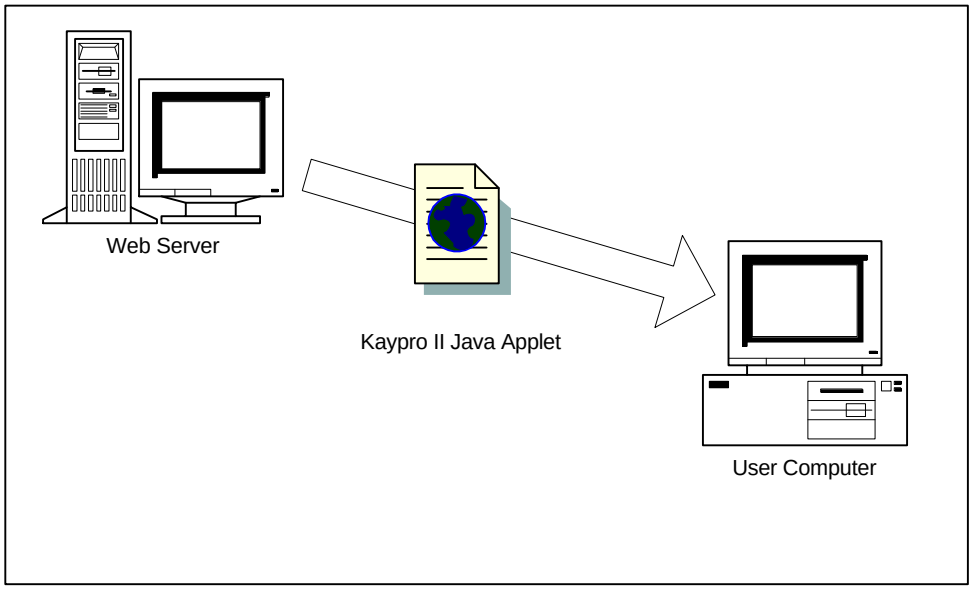


Figure 2 Kaypro II Emulator Java Applet Transfer

The Kaypro II emulation shall be implemented as a Java applet. Java applets reside on remote servers (see Figure 2). When a user browses an HTML web page containing reference to the emulator applet, the applet byte code is transferred to the User's computer and executed[1].

- The Kaypro II emulation shall be implemented in Java
- The implementation shall be pure Java (e.g. No Microsoft extensions).
- The Kaypro II emulation shall be implemented as an applet.
- The Kaypro II emulation shall be made available via the world wide web
- The Java interface shall be kept minimal, to afford quicker load times
- The Implementation shall use JDK 1.1.6

[1] See Sun Micro's description of the Java environment and language.

2.2.4 User Interface

Class Kaypro is a Task

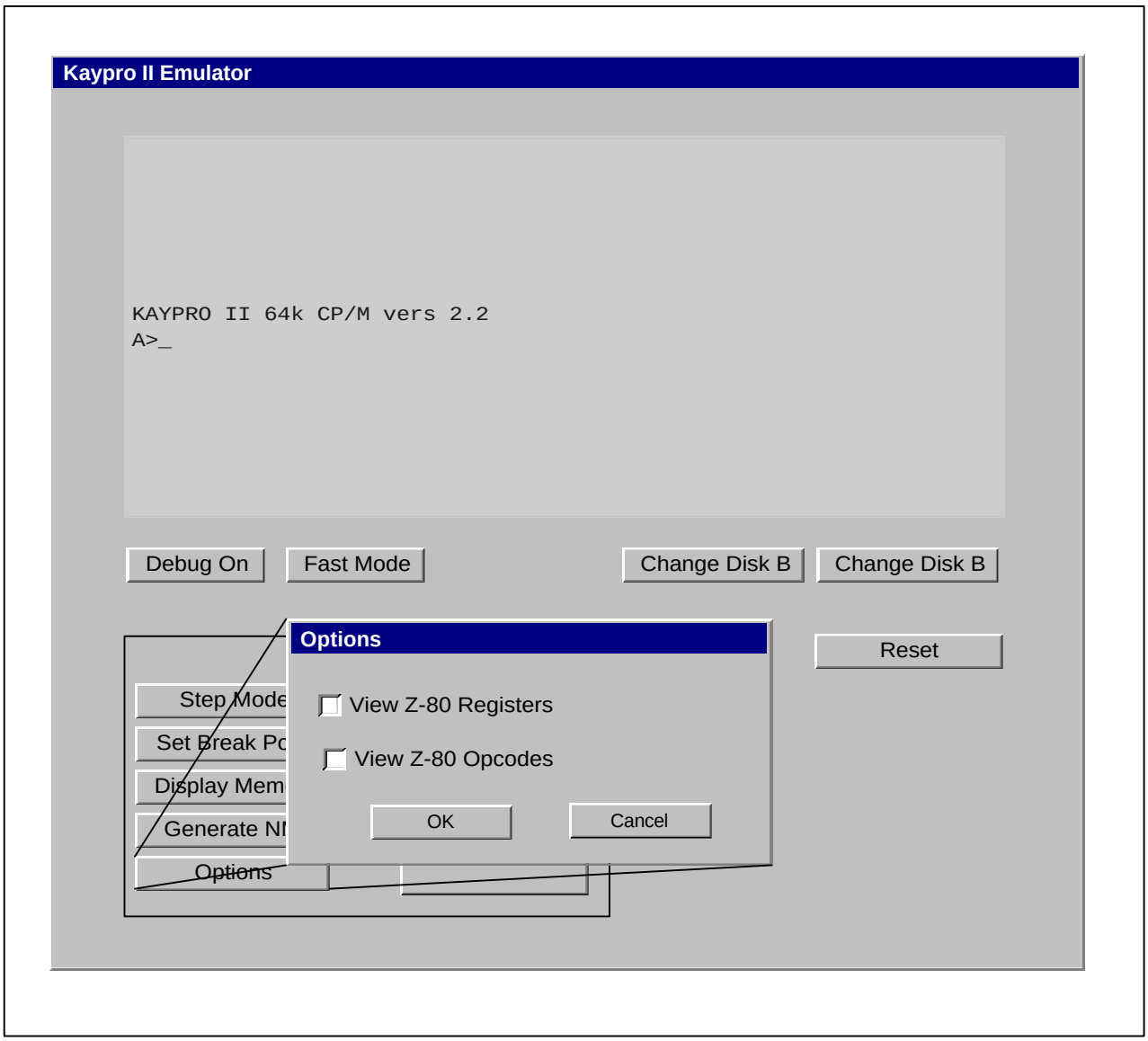


Figure 3, User Interface Example

2.2.4.1 User Interface Members

Member	Scope	Description	Type
cpu	Public	Instantiation of the cpu object.	CPU
hardware	Public	Instantiaiton of the hardware object.	HARDWARE
screen	Public	Instantiation of the screen object.	Screen
floppy	Public	Instantiation of the floppy object.	Floppy
sio	Public	Instantiation of the sio object.	SIO

pio	Public	Instantiation of the pio object.	PIO
printer	Public	Instantiation of the printer.	Printer
syspio	Public	Instantiation of the syspio.	SysPIO
uimonitor	Public	Instantiation of the uimonitor.	UIMonitor
tracker	Public	Instantiation of the mediatracker.	MediaTracker
KayproCanvas	Public	Instantiation of the ImageCanvas. Used for the spinning Kaypro.	ImageCanvas
Logo	Public	Instantiation of the LogoCanvas. Used for the Kaypro II Logo.	LogoCanvas
OutputCanvas	Public	Instansiation of the OutCanvas. Used for the real mode output.	OutCanvas
MainScreen	Private	Main panel that holds all of the buttons and debug panel.	Panel
BDebugMode	Private	Button that toggles the debug mode on or off.	Button
BFastMode	Private	Button that toggles between the fast and real graphics mode.	Button
BReset	Private	Button that resets the Kaypro II emulation.	Button
BChangeA	Private	Button that brings up a dialog box that asks for the new virtual disk for drive A.	Button
BChangeB	Private	Button that bring up a dialog box that asks for the new virtual disk for drive B.	Button
Debug	Private	Panel that holds the debug buttons. Belongs to MainScreen panel.	Panel
Ldebug	Private	Label for the debug mode panel.	Label
BStepMode	Private	Button that toggles between step and run modes.	Button
BBreakpoint	Private	Button that brings up a dialog box that asks the user what the breakpoint should be.	Button
BMemoryDump	Private	Button that brings up a dialog box that asks the user what memory they want to view.	Button
BInterrupt	Private	Button that sends a NMI interrupt to the hardware object.	Button
BOptions	Private	Button that brings up a dialog box that asks the user if they want to view Registers and/or Opcodes in debug mode.	Button
BSingleStep	Private	Button that sends a step command to the cpu object.	Button
OptionDialog	Private	Panel for a dialog box that asks the user for the view options.	Panel
LOptionDialog	Private	Label for the option dialog panel.	Label
CViewReg	Private	Checkbox that sets the view registers flag to see the registers when debugging.	Checkbox
CViewOp	Private	Checkbox that sets the view opcodes flag to see the opcodes when debugging.	Checkbox

BOptionsOK	Private	Button that exits the OptionDialog panel and sets the view registers and view opcodes flags accordingly.	Button
BOptionsCancel	Private	Button that exits the OptionsDialog panel without changing the view registers and view opcodes flags.	Button
ChangeDialogA	Private	Panel for the dialog box that asks the user for the location of the virtual disk for virtual disk drive A.	Panel
LChangeDialogA	Private	Label for the Change Disk A panel.	Label
TChangeA	Private	The textbox that the user type in the location and name of the virtual disk being loaded.	Textbox
LChangeA	Private	Label: “Location of the virtual disk:”	Label
BChangeAOK	Private	Button that exits the ChangeDialogA panel changing the virtual disk A to the selected virtual disk in the TChangeA textbox.	Button
BChangeACancel	Private	Button that exits the ChangeDialogA panel without changing the virtual disk A.	Button
ChangeDialogB	Private	Panel for the dialog box that asks the user for the location of the virtual disk for virtual disk drive B.	Panel
LChangeDialogB	Private	Label for the Change Disk B panel.	Label
TChangeB	Private	The textbox that the user type in the location and name of the virtual disk being loaded.	Textbox
LChangeB	Private	Label: “Location of the virtual disk:”	Label
BChangeBOK	Private	Button that exits the ChangeDialogB panel changing the virtual disk B to the selected virtual disk in the TChangeB textbox.	Button
BChangeBCancel	Private	Button that exits the ChangeDialogB panel without changing the virtual disk B.	Button
Breakpoint	Private	Panel for the dialog box that asks the user for the breakpoint.	Panel
LBreakpointD	Private	Label for the breakpoint panel.	Label
TBreakpoint	Private	The textbox that the user types in the location in memory they want the emulation to break at.	Textbox
LBreakpoint	Private	Label: “Location in memory to break at:”	Label
BBreakpointOK	Private	Button that exits the Breakpoint dialog box and set the breakpoint that is in the TBreakpoint textbox.	Button
BRemoveBreak	Private	Button that exits the Breakpoint dialog box and removes the current breakpoint from the CPU.	Button
BBreakpointCancel	Private	Button that exits the Breakpoint	Button

		dialog box without setting a breakpoint.	
MemoryDump	Private	Panel for the dialog box that asks the user the start and end location for a memory dump.	Panel
LMemoryDump	Private	Label for the memory dump panel.	Label
TStartDump	Private	Textbox that the user inputs the start location of the memory dump.	Textbox
LStartDump	Private	Label: “Start location of the memory dump:”	Label
TEndDump	Private	Textbox that the user inputs the end location of the memory dump.	Textbox
LEndDump	Private	Label: “End location of the memory dump:”	Label
LMemoryBank	Private	Label: “Select memory bank:”	Label
CBGMemoryBank	Private	This is a checkbox group for the user to select a memory dump from RAM, ROM, or Video memory.	CheckboxGroup
CRAM	Private	When selected the RAM is displayed for the memory dump. Part of CBGMemoryBank CheckboxGroup.	Checkbox
CROM	Private	When selected the ROM is displayed for the memory dump. Part of CBGMemoryBank CheckboxGroup.	Checkbox
CVideo	Private	When selected the Video memory is displayed for the memory dump. Part of CBGMemoryBank CheckboxGroup.	Checkbox
BDumpOK	Private	Button that exits the MemoryDump dialog box showing the memory from the value in the TStartDump textbox to the TEndDump textbox.	Button
BDumpCancel	Private	Button that exits the MemoryDump dialog box without showing any memory locations.	Button
ErrorPanel	Private	Panel for the error dialog box. Used when the user puts in wrong values into textboxes.	Panel
LError	Private	Label: “Error in input: please re-enter value.”	Label
BErrorOK	Private	Button that exits the ErrorPanel.	Button
LVersion	Private	Label that shows the version of the emulator.	Label
FPrinterScreen	Private	Frame for the printer output.	Frame
TAPrinter	Private	TextArea for the output from the PIO chip.	TextArea
FViewRegOp	Private	Frame for the registers and opcodes output.	Frame
TARegOp	Private	TextArea for the registers and opcodes output from the cpu.	TextArea
FMemory	Private	Frame for the memory dump.	Frame

TADump	Private	TextArea for the memory dump.	TextArea
ViewRegisters	Private	Used to determine if the registers should be output. TRUE means output registers, FALSE means don't output registers.	boolean
ViewOpcodes	Private	Used to determine if the opcodes should be output. TRUE means output opcodes, FALSE means don't output opcodes.	boolean
VirtualLocation	Private	String variable to hold the user input in the Change Disk A and Change Disk B dialog boxes.	String
MemoryStart	Private	String variable to hold the user input for the start location of a memory dump. Used for the memory dump dialog.	String
MemoryEnd	Private	String variable to hold the user input for the end location of a memory dump. Used for the memory dump dialog.	String
MemoryBank	Private	Variable to hold the memory bank selected for a memory dump. 0 means dump the RAM, 1 means dump the ROM, and 2 means dump the Video RAM.	int
DrawMode	Private	Variable to hold the drawing mode. 0 means draw in fast mode, and 1 means draw in real mode.	int
SetBreakpoint	Private	String variable to hold the input from the user in the Breakpoint dialog. This is the location for the breakpoint.	String
DialogUp	Private	Variable to hold the state of the dialog panels that are being shown. This makes the panels modal, so buttons below the panel cannot be pressed.	boolean
ErrorDialogUp	Private	Variable to hold the state of the error dialog that is shown. This makes the error panel modal, so buttons below the panel cannot be pressed.	boolean
image	Private	Variable to hold the Kaypro logo image.	Image[]
KEYPORTDATA	Private	Holds the port for keyboard data input.	short
KEYPORTCONTROL	Private	Holds the port for keyboard control input.	short

2.2.4.2 Methods

Method	Scope	Parameter Values	Return Values	Description
main	Public	String[]	void	Creates the main applet frame.
init, jbInit	Public	void	void	Initializes the emulation; instantiates CPU and Hardware objects and all GUI objects.
start	Public	void	void	Starts the emulation applet.
stop	Public	void	void	Destorys and cleans up the applet.
run	Public	void	void	Runs the Kaypro II emulation.
paint	Public	Graphics g	void	Repaints the screen in real mode or fast mode according to the mode.
destroy	Public	void	void	Set the visibility of all frames to false.
getAppletInfo	Public	void	String	Returns the name of the applet.
getParameterInfo	Public	void	String[]	Returns the parameter information sent to the applet.
getParameter	Public	String, String	String	Sets and returns the parameters sent to applet.
BDebugMode_actionPerformed	Private	ActionEvent e	void	Handles the debug mode button click.
BFastMode_actionPerformed	Private	ActionEvent e	void	Handles the fast/real mode button click.
BReset_actionPerformed	Private	ActionEvent e	void	Handles the reset button click.
BChangeA_actionPerformed	Private	ActionEvent e	void	Handles the change a button click.
BChangeB_actionPerformed	Private	ActionEvent e	void	Handles the change b button click.
BStepMode_actionPerformed	Private	ActionEvent e	void	Handles the step mode button click.
BBreakpoint_actionPerformed	Private	ActionEvent e	void	Handles the set breakpoint button click.

BMemoryDump_actionPerformed	Private	ActionEvent e	void	Handles the memory dump button click.
BInterrupt_actionPerformed	Private	ActionEvent e	void	Handles the generate interrupt button click.
BOptions_actionPerformed	Private	ActionEvent e	void	Handles the view options button click.
BSingleStep_actionPerformed	Private	ActionEvent e	void	Handles the single step button click.
BDumpOK_actionPerformed	Private	ActionEvent e	void	Handles the OK button click in the MemoryDump dialog.
BDumpCancel_actionPerformed	Private	ActionEvent e	void	Handles the Cancel button click in the MemoryDump dialog.
BBreakpointOK_actionPerformed	Private	ActionEvent e	void	Handles the OK button click in the Breakpoint dialog.
BBreakpointCancel_actionPerformed	Private	ActionEvent e	void	Handles the Cancel button click in the Breakpoint dialog.
BOptionsOK_actionPerformed	Private	ActionEvent e	void	Handles the OK button click in the OptionDialog dialog.
BOptionsCancel_actionPerformed	Private	ActionEvent e	void	Handles the Cancel button click in the OptionDialog dialog.
BChangeAOK_actionPerformed	Private	ActionEvent e	void	Handles the OK button click in the ChangeDialogA dialog.
BChangeACancel_actionPerformed	Private	ActionEvent e	void	Handles the Cancel button click in the ChangeDialogA dialog.
BChangeBOK_actionPerformed	Private	ActionEvent e	void	Handles the OK button click in the ChangeDialogB dialog.
BChangeBCancel_actionPerformed	Private	ActionEvent e	void	Handles the Cancel button click in the ChangeDialogB dialog.
BErrorOK_actionPerformed	Private	ActionEvent e	void	Handles the OK button click in the Error dialog.

BRemoveBreak_actionPerformed	Private	ActionEvent e	void	Handles the Remove button click in the Breakpoint dialog.
OutputPanel_keyTyped	Private	KeyEvent e	void	Handles the key typed event in the OutputPanel. This is were all keyboard input is captured by the emulator.

2.2.5 ImageCanvas extends Canvas

Image canvas provides a canvas on which a spinning picture of the Kaypro II computer is loaded. The images allow the user to view a 3-D rotation of an actual Kaypro II computer.

2.2.5.1 ImageCanvas Members

Member	Scope	Description	Type
kaimages	Private	Images used for the animation.	Image[]
thread	Private	Thread object to make this object a thread.	Thread
tracker	Private	MediaTracker object to track the loading of the images.	MediaTracker
currentimg	Private	Variable to hold the current image to be shown.	int

2.2.5.2 ImageCanvas Methods

Method	Scope	Parameter Values	Return Values	Description
init	Public	void	void	Initializes all objects needed for this object. Also loads the images into the MediaTracker.
paint	Public	Graphics g	void	Paint method for this object to paint the images onto the canvas object.
run	Public	void	void	This method runs this thread. As soon as the images are loaded the images are cycled forever.
start	Public	void	void	Starts the thread so this object will run with the rest of the emulation.
stop	Public	void	void	Stop the thread.

2.2.6 LogoCanvas extends Canvas

2.2.6.1 LogoCanvas Members

The logo canvas provides an object on which the Kaypro II logo is located. The Kaypro II logo was scanned from the original Kaypro II, and stored as a bitmap. The bitmap is loaded when the emulation is started.

Member	Scope	Description	Type
image	Private	Image to be sent to this object and shown.	Image
clear	Private	Variable to tell this object to clear the Canvas before drawing the image.	boolean

2.2.6.2 LogoCanvas Methods

Method	Scope	Parameter Values	Return Values	Description
Paint	Public	Graphics g	void	Paint method for this object to paint the image onto the canvas object.
SetImage	Public	Image image	void	Sets the image to be shown onto the canvas.
Update	Public	Graphics g	void	Calls the paint method each time this object is called or the screen needs to repainted.

2.2.7 OutCanvas extends Canvas

2.2.7.1 OutCanvas Members

Out Canvas holds the Real mode image. The real mode image allows the user to view the actual character set of the original Kaypro II computer.

Member	Scope	Description	Type
Image	Private	Image to be shown using MemoryImageSource. Image will be rendered using this object.	Image
Hardware	Private	Hardware object pointer to get the video RAM.	HARDWARE
CharSet	Private	Array that holds the character rom from the original Kaypro II computer.	Byte[]
ColorModel	Private	This object holds the color to be used when rendering the image.	ColorModel
Pixels	Private	Array that holds the pixels that are	Byte[]

		turned on or off when the image is rendered.	
Rscreen	Private	MemoryImageSource that will render the image used.	MemoryImageSource

2.2.7.2 OutCanvas Methods

Method	Scope	Parameter Values	Return Values	Description
Paint	Public	Graphics g	void	Paint method for this object to paint the image onto the canvas object.
Update	Public	Graphics g	void	Calls the paint method each time this object is called or a repaint is needed.
realUpdate	Public	Graphics g	void	This method is called externally to speed up the drawing. The drawing faster by eliminating the call to paint.
generateColorModel	Public	void	void	This method creates the color model used by the MemoryImageSource. Only two colors are needed because the Kaypro II screen is monochrome.
generatePixels	Public	void	void	This method mathematically fills the pixels array to render the image.
MakeScreenImage	Public	void	void	This method creates the MemoryImageSource and sets it to an animated MemoryImageSource.

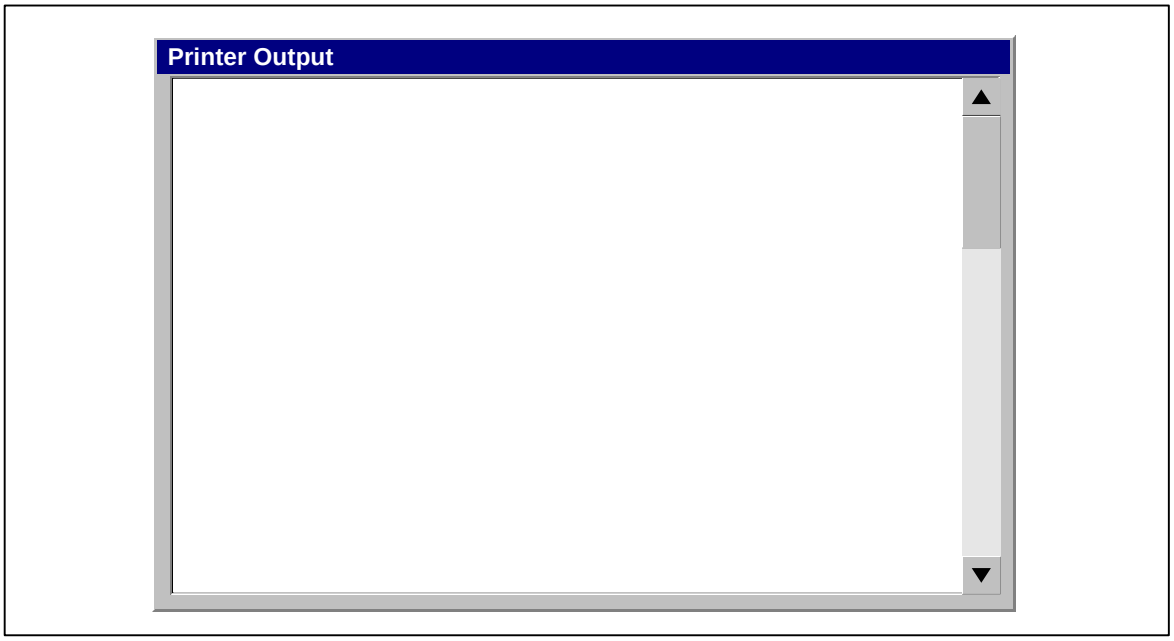


Figure 4 Output Screen Example

2.2.8 Z-80 CPU Emulation

- The Kaypro II utilizes the Z-80 Processor. The CPU emulated shall be the Z-80
- The CPU instruction set shall conform to those presented in the *Zilog Z80 Microprocessor Family User's Manual, Part number Q1/95 DC 8309-1.*

2.2.8.1 Z-80 Features

The Z-80 CPU contains a number of notable features. These include:

- One 8-bit Accumulator
- One 8-bit flags register
- Six 8-bit general purpose register, that can be mapped to three 16-bit registers
- An alternate register set
- An interrupt vector register
- Two 16-bit index registers
- One stack pointer
- One program counter

2.2.8.2 Z-80 Instruction Set

The Z-80 instructions are a superset of the 8080.
A summary of the Z-80 instruction set is included in appendix A.

The Z-80 instruction set consists of the following groups of operations:

- Load and exchange
- Block transfer and search
- Arithmetic and Logical
- Rotate and shift

- Bit Manipulation
- Jump, Call and return
- Input and Output
- CPU Control (NOP, HALT, etc)
- The RLA command as documented in the Zilog user's manual contains an error in the rotate diagram. The emulation shall support shift left, whereas the manual depicts shift right.

2.2.8.3 Z-80 Arithmetic Logic Unit (ALU)

The Z-80 ALU supplies the following functions:

- Add
- Subtract
- Logical AND
- Logical OR
- Logical Exclusive OR
- Compare
- Shift and rotate
- Increment and decrement
- Bit operations

- The ALU shall be implemented as a function of the CPU. It may not be implemented separately.

2.2.8.4 Z-80 Addressing modes

The Z-80 CPU shall support the following addressing modes:

- Immediate, where data is explicitly specified within the instruction (8 bit)
- Immediate extended, where data is specified within the instruction (16 bit)
- Zero page, where a single byte instruction may call one of eight zero-page locations
- Relative, where the following byte specifies a relative address
- Extended, where a 16 bit value specifies the location of an indirect address
- Indexed, where an index register and an offset specify an absolute address
- Register addressing, where a particular register specifies an address location
- Implied addressing, where the opcode automatically implies a CPU register
- Register indirect addressing, where the register contains an indirect address reference
- Bit addressing, where memory or registers may directly manipulate individual bits

2.2.8.5 Main Registers

The Z-80 emulation shall include the following main registers. These registers shall be contained within the processor, and be accessible to the CPU instructions. The alternate instructions are accessible via a Z-80 swap command. This command swaps the main and alternate register sets. Unless swapped, the alternate register set is not accessible to the Z-80 instructions.

Main Register Set		**Alternate Register Set**	
Accumulator A	Flags F	Accumulator A'	Flags F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

Table 1, Z-80 Primary Register Set

2.2.8.6 Special Purpose Registers

Z-80 special purpose registers shall be included in the emulation. These registers assist in indirect addressing via Z-80 instructions, specifically, the IX and IY index registers. The program counter controls program execution, while the stack pointer register controls stack operations. The Z-80 CPU includes instructions for stack manipulation.

Special Purpose Registers	
Interrupt Vector I	Memory Refresh Register R
Index Register IX	
Index Register IY	
Stack Pointer SP	
Program Counter PC	

Table 2, Z-80 Special Purpose Register Set

- The Interrupt vector, I, is used for mode 2 interrupts (described below). It shall be implemented.
- The Memory refresh register is used for dynamic memory refresh. It is incremented each time an instruction is executed. Dynamic memory is a function of hardware implementation, and is not needed. The memory refresh register, R, shall not be implemented.

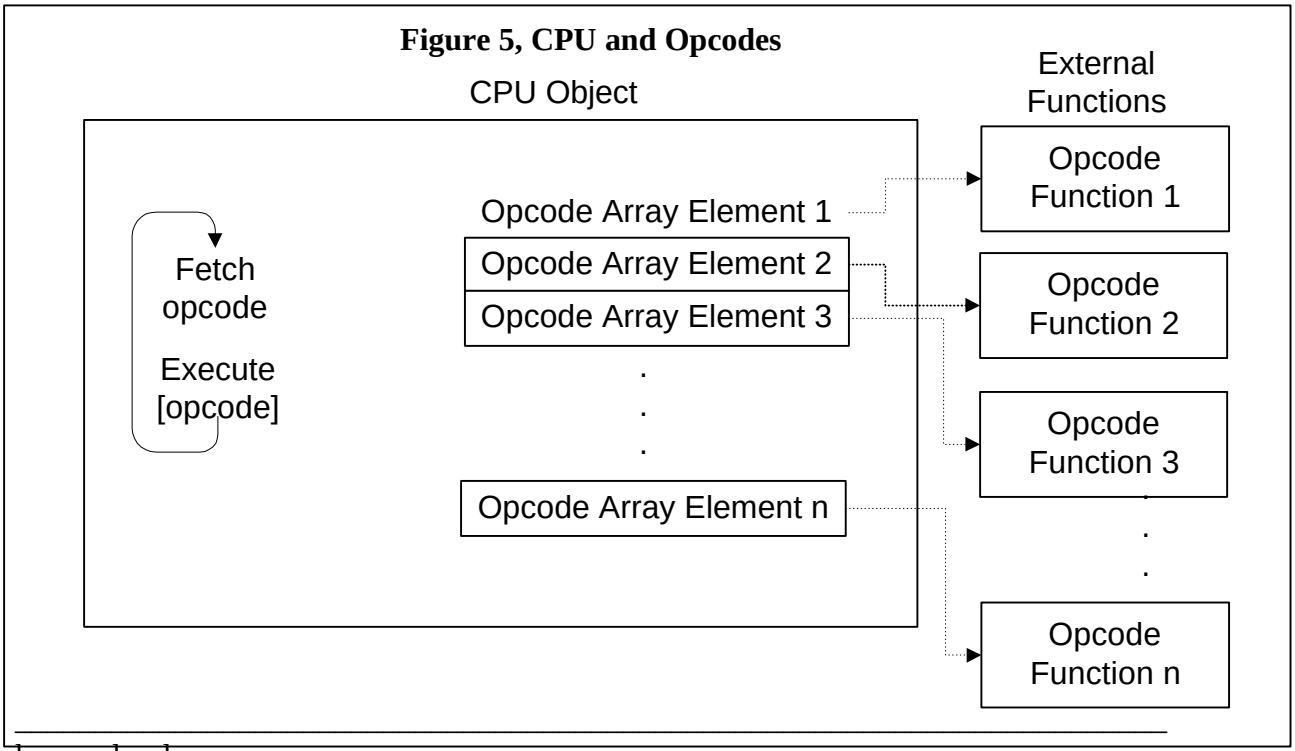
2.2.8.7 Bus Timing and Signals

Bus timing and associated signals shall not be implemented. Only the function of the CPU shall be implemented. Exact CPU speed shall not be governed, except within the limits of the executing hardware and software.

2.2.8.8 CPU Object

CPU Object is a Task

Figure 5, CPU and Opcodes



The CPU shall be implemented as an object. It shall have access via member functions. The CPU object shall contain an array of opcode references. The CPU shall fetch an opcode, and use that value as an index into the array of object references.

- The Z-80 CPU shall be implemented as an object.
- Each instruction may be implemented as a separate object or as one of a group of objects
- Each instruction may be accessed via an array of opcode references contained within the CPU object
- The Z-80 object shall be implemented as a Java class
- The Z-80 object shall be implemented as a Java thread
- The CPU object shall contain certain necessary private variables and public functions. The variables shall serve as internal registers, flags, etc.
- The public functions shall allow external objects to access the internal CPU operations.

2.2.8.8.1 CPU Object Data Members

Name	Scope	Type	Description
StepMode	Private	Boolean	The CPU shall be capable of operating in two modes: debug and run. In debug mode, the CPU shall execute one instruction at a time. The StepMode variable shall control the CPU mode. 0=step mode off, 1=step mode on
hHardware	Private	Hardware	The CPU shall have access to the hardware object. The hardware variable shall be a reference to the hardware object. Reference to hardware object. Needed to read and write data to/from object
SP	Private	Short	Stack pointer
PC	Private	Short	Program Counter
IX	Private	Short	Index register
IY	Private	Short	Index register
A	Private	Short	A register
B	Private	Short	B register
C	Private	Short	C register
D	Private	Short	D register
E	Private	Short	E register
H	Private	Short	H register
L	Private	Short	L register
A1	Private	Short	A register (alternate set)
B1	Private	Short	B register (alternate set)
C1	Private	Short	C register (alternate set)
D1	Private	Short	D register (alternate set)
E1	Private	Short	E register (alternate set)
H1	Private	Short	H register (alternate set)
L1	Private	Short	L register (alternate set)
R	Private	Short	Memory refresh register
I	Private	Short	Interrupt control vector
IFF	Private	Boolean	Interrupt enable
Halt	Private	Boolean	The CPU contains a command to halt the processor. When the CPU is in halt state, the Halt variable shall indicate its state. State of processor (0=running, 1=halted)
Reset	Private	Boolean	The CPU may be reset. The Reset variable contains the current reset state of the CPU State of processor (0=running, 1=reset)
Break	Private	Int	A breakpoint may be set. This indicates at what address the CPU will break. The break address is held in the Break variable. PC of breakpoint
Cmd[]	Private	Opcode	Array of opcode objects. Each points to a single Z-80 opcode. *See opcode object*.

Table 3, CPU data Members

2.2.8.8.2 CPU Member Function Summary

Member	Scope	Parameters	Return Value	Description
SetBreakPoint	Public	Short Address (0-0xffff)	Short 0 = Breakpoint set 1 = Breakpoint error	Sets a CPU breakpoint
GetStepMode	Public	None	boolean false = Run mode true = Step Mode	Returns the current status of Step/Mode
SetStepMode	Public	boolean true = Run Mode false = Step Mode	None	Sets step mode to run or step
Registers2String	Public	None	String Text string containing current register status	Returns text string containing CPU register status. Used for debugging
Flags2String	Public	None	String Text string containing current flag status	Returns text string containing CPU flag status. Used for debugging
Opcode2String	Public	None	String Text String containing current opcode mnemonic an parameters	Returns text string containing CPU opcode mnemonic. Used for debugging
Step	Public	None	Void	Single steps CPU one instruction. CPU must be in step mode
Start	Public	Hardware Reference to hardware object	Void	Starts the CPU thread
Busy	Public	None	Boolean True=Executing step False=Waiting for step (ok to read opcodes, registers or flags)	Valid when in step mode only. Returns state of CPU. True if CPU is busy, False if CPU is idle. External routines should not try to read flags opcodes, or registers while CPU is busy. erroneous results may occur.
CPU	Public	None	Void	CPU Constructor

Table 4, CPU Public Functions

2.2.8.8.3 CPU Public Member Function Descriptions

2.2.8.8.3.1 SetBreakPoint Public Member Function

The CPU object supports a single breakpoint. The SetBreakPoint function sets the address on which the CPU will enter a break mode. The CPU shall break when an opcode is fetched from the given address. A break will not be performed on a data read or write.

When the CPU encounters a breakpoint, it will place itself in step mode, and stop executing commands.

2.2.8.8.3.2 GetStepMode Public Member Function

The GetStepMode function returns the current state of the CPU. Possible states are:

- Step mode
- Run Mode

2.2.8.8.3.3 SetStepMode Public Member Function

The SetStepMode function allows external entities to manually set the mode of the CPU. Possible modes are:

- Step mode
- Run Mode

2.2.8.8.3.4 RegisterDisplay Public Member Function

The register display function returns a string containing the current register status. An example would be:

```
PC=579 A=0 BC=7f DE=e406 HL=2d IX=0 IY=0 SP=fbfc
```

2.2.8.8.3.5 OpcodeDisplay Public Member Function

The register display function returns a string containing the current register status. An example would be:

```
JR Z,57d
```

2.2.8.8.3.6 FlagDisplay Public Member Function

The register display function returns a string containing the current register status. An example would be:

```
S=0 Z=1 H=1 PV=1 N=0 C=0
```

2.2.8.8.3.7 Step Public Member Function

Active only when CPU is in step mode. Executes one complete instruction.

2.2.8.8.3.8 Start Public Member Function

2.2.8.8.4 External Hardware Object Access

The CPU objects shall have access to public functions and public data within the hardware object. The CPU shall be able to perform the following functions on the hardware object:

2.2.8.8.4.1 Short Readport (short port)

Reads a byte from a specific hardware port. The port is passed as the only parameter.

2.2.8.8.4.2 Void Writeport (short port, short data)

Writes a byte to a specific hardware port. The port number is passed as the first parameter, while the data is passed as the second parameter.

2.2.8.8.4.3 Int ReadWord (int address)

Reads a word from memory. The address to read from is the only parameter. The read is context dependent. That is, the read will occur from the currently selected bank.

The return value is expected to be in the proper order. The Z-80 stores the LSB first, and the MSB second. Thus, the return value is expected to be *LSB+MSB*256.*

2.2.8.8.4.4 Void WriteWord (int address, int data)

Writes a word to memory. The address to read from is the first parameter, while the data to be written is the second parameter. The write is context dependent. That is, the write will occur from the currently selected bank.

The value is expected to be stored in the proper order. The Z-80 stores the LSB first, and the MSB second. Thus, the stored value is expected to be *LSB+MSB*256.*

2.2.8.8.4.5 Short ReadByte (int address)

Reads a byte from memory. The address to read from is the only parameter. The read is context dependent. That is, the read will occur from the currently selected bank.

2.2.8.8.4.6 Void WriteByte (int address, int data)

Writes a byte to memory. The address to read from is the first parameter, while the data to be written is the second parameter. The write is context dependent. That is, the write will occur from the currently selected bank.

2.2.8.8.4.7 boolean ResetStatus ()

Reads reset state from hardware object.
The Z-80 CPU shall support implementation of the RESET function. When reset, the CPU shall force a jump to location 0x00 upon completion of the current command.

2.2.8.8.4.8 boolean NMIStatus()

Reads NMI state from hardware object
The Z-80 supports one non-maskable interrupt. This interrupt is executed when the NMI function of the hardware object returns true. The NMI interrupt forces a CPU restart (call) to location 0x66 upon the completion of the current instruction.

- The non-maskable interrupt may not be disabled.

2.2.8.8.4.9 boolean IntStatus ()

The Z-80 CPU supports 3 modes of maskable interrupts: mode 0, mode 1, and mode2. The int function accepts a value from the hardware object. This value is used as an offset for modes 0 and 2. This value is not implemented in mode 1.

- Maskable interrupts may be disabled via CPU instruction.

2.2.8.8.4.9.1 Mode 0

An interrupt is executed when the INT line of the CPU is activated. The INT mode 0 interrupt forces execution of the instruction placed on the bus by the interrupting device. The execution of the device-supplied instruction takes place upon the completion of the current instruction.

- Mode 0 shall be implemented.

2.2.8.8.4.9.2 Mode 1

An interrupt is executed when the INT line of the CPU is activated. The INT mode 1 interrupt forces a CPU restart (call) to location 0x38 upon the completion of the current instruction.

- Mode 1 shall be implemented.

2.2.8.8.4.9.3 Mode 2

An interrupt is executed when the INT line of the CPU is activated. The INT mode 2 requires that the programmer setup a table of 16 bit service routine addresses. When an interrupt is generated, a 16 bit address is created. This address points to an element in the table.

The upper 8-bits of the address is specified by the programmer, and stored in the I register. The lower 8-bits are supplied by the interrupting device. This address is used by the CPU to index into the programmer-supplied table. The index points to the address of the interrupt service routine.

The execution of the interrupt service routine takes place upon the completion of the current instruction.

- Mode 2 shall be implemented.

2.2.8.8.4.10 Int GetINTVector ()

Returns the vector from the INT (modes 0 and 2)

2.2.8.9 Opcode Object

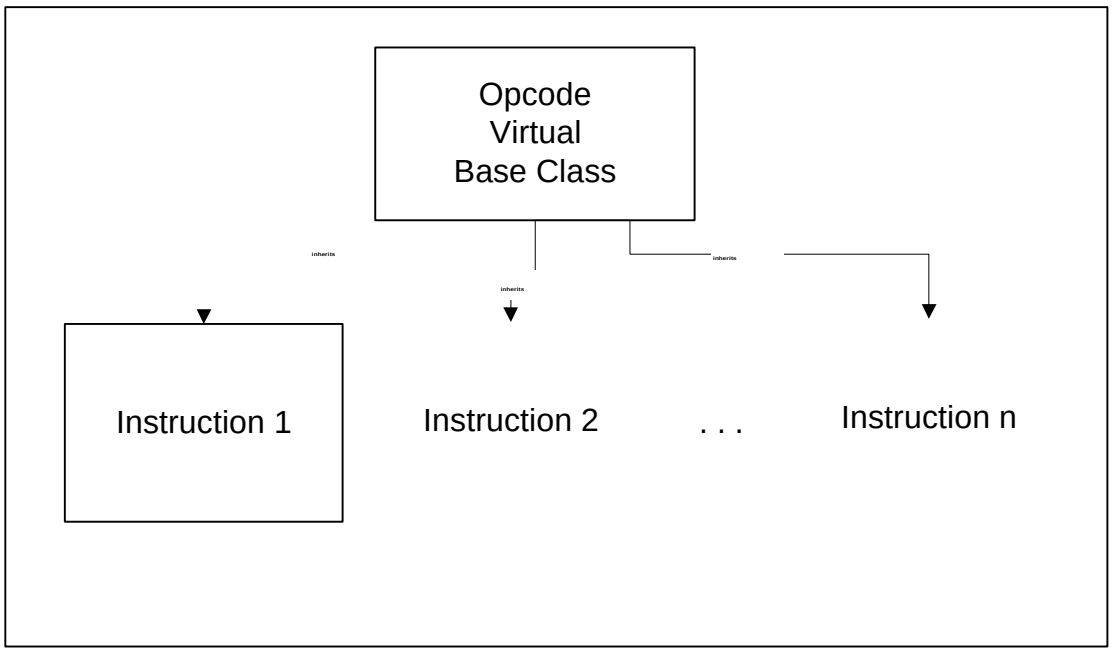


Figure 6, Opcode Object

The opcode object is a virtual base class. Its purpose is to allow additional classes to inherit its two basic functions.

The opcode base class contains two virtual functions.

- Execute
- Log

The rules of inheritance allow opcodes to point to instructions. For example, the following code would be implemented for each instruction (or groups of instructions).

```
class instruction1 extends opcode{
        public void log(){
                //Log operation
        }

        public void execute(){
                //Execute command
        }
}
```

The instruction1 class would extend the functionality of the base class. It would supply the functionality for the log and execute functions. Each instruction, or group of instructions will have a similar definition.

Within the CPU class, the following code would be implemented.

```
private opcode cmd[] = new opcode[maxopcodes];
opcode[0] =  new instruction1();
opcode[1] =  new instruction2();
opcode[2] =  new instruction3();
```

This code allows each instruction to be referenced via an array index. Executing an instruction is then quite simple. It would be accomplished as follows.

```
While(true){
        Index=fetch();
        Opcode[Index].execute();
        Opcode[Index].log();
}
```

The CPU object simply fetches the next opcode, then uses that opcode as an index into an array of opcode pointers. It uses the pointer to execute the correct opcodes member function(s).

2.2.8.9.1 Opcode Object Data Members

There are no required opcode data members

2.2.8.9.2 Opcode Member Function Summary

Function	Parameters	Return Value	Description
Log	None	String containing opcode string	Returns a string containing text of the command executed. For example, the return string may contain: *JR Z, 57d* This function is provided for debugging purposes.
Execute	None	None	Executes the current instruction, and sets the PC to the next instruction.

2.2.8.9.3 External CPU Object Access

The opcode object shall have access to functions and data within the CPU object. Because the two are intimately tied to each other they shall be friend functions.

2.2.9 Object Hardware

Hardware is a Task

Figure 7, Port Emulation, Kaypro II functional view, shows the Kaypro II architecture. Central to the architecture is the hardware. The hardware represents the communication mechanism between the various system components. The hardware represents discrete chips and connection logic within the actual Kaypro II. The hardware shall be a link from the CPU to all the other devices. The hardware connects the following devices.

- Keyboard
- Screen
- Memory (Ram/Video/RAM)
- Bank switching
- I/O Ports
- Disk Drives
- Motor control, indicator lights, etc

The hardware passes the following signals (messages):

- User action
- NMI
- INT
- Reset
- Read memory
- Write memory
- Port commands

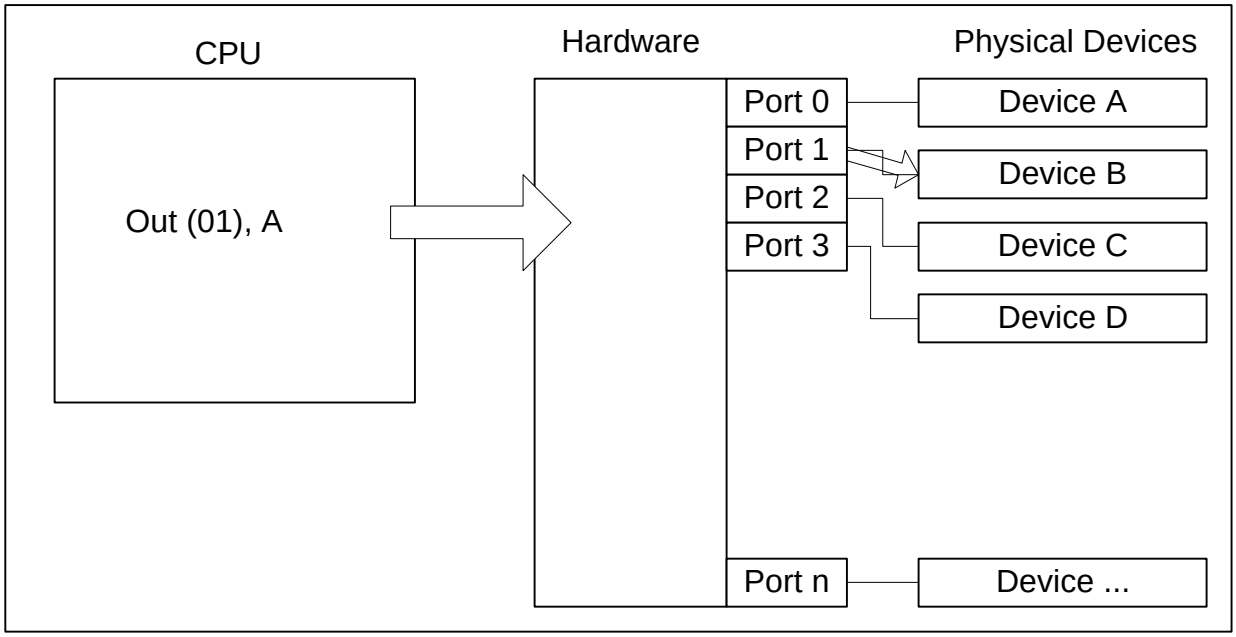


Figure 7, Port Emulation

2.2.9.1 Memory Emulation

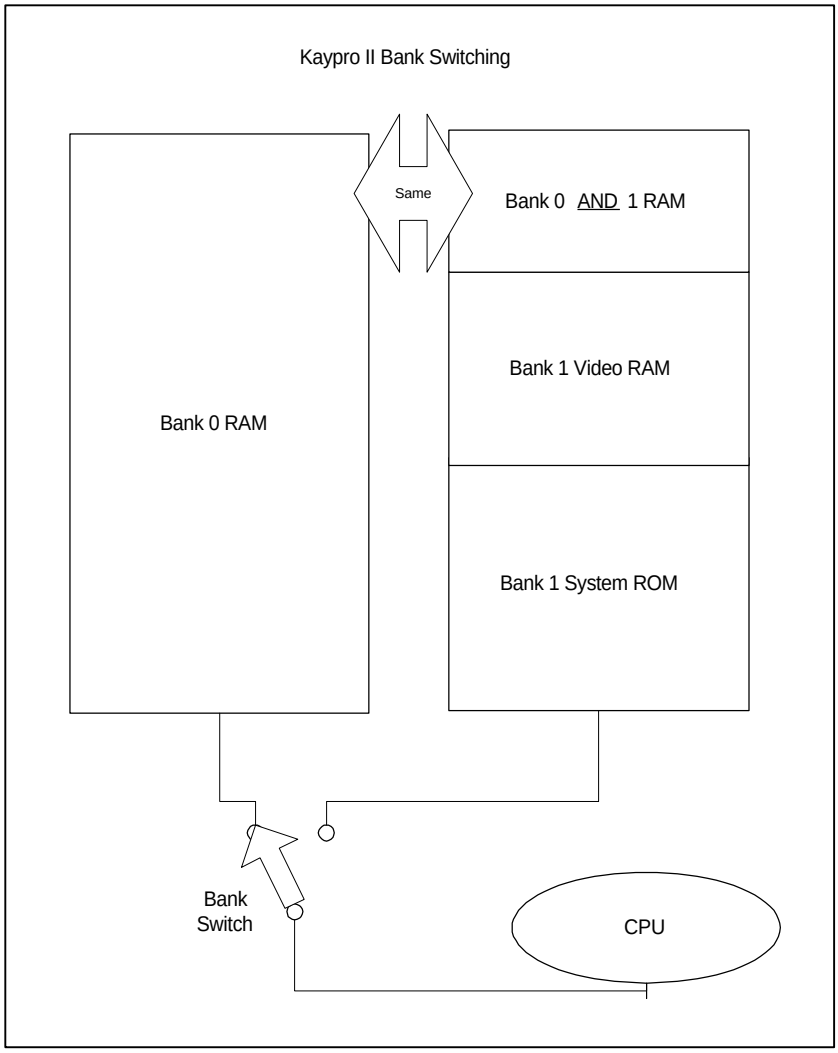


Figure 8 Kaypro II Bank Switching

The Kaypro II utilizes two banks of memory. Bank 0 contains 64K of linear RAM. Bank 1 contains the video and system ROM. Notice from Figure 8 that bank 0 and bank 1 share high memory.

One way to understand the Kaypro II banking scheme is to visualize a switch (see Figure 8). The CPU executes instructions from memory. The memory that the CPU sees is determined by the bank switch. Depending on the position of the bank switch, the CPU will operate on data from either bank 0 or bank 1. The bank switch is thrown electronically. This allows the Kaypro II to support 64K programs, while still supporting memory mapped video, and bootstrap ROM.

Note that the upper portion of bank 0 and bank 1 share bank 0's RAM. This allows a single program to operate in both memory domains.

Bank 1 contains ROM as well as video. The ROM, referred to as the "System ROM," is contained within a 2716 EPROM. This EPROM is pre-programmed with utility routines, as well as a bootstrap loader. When

the system is reset, bank 1 is selected, and code from within the System ROM is executed. This code loads the operating system from floppy and initializes the system. The complete memory map for the Kaypro II is shown in Table 5.

Bank	Type	Range
0	System RAM	0x0000 - 0xFFFF
1	System ROM (2716)	0x0000 – 0x2FFF
1	Video RAM	0x3000 – 0x3FFF
1	System RAM[2]	0x4000 – 0xFFFF

Table 5, Kaypro II Memory Map

- The emulated RAM shall be 64K bytes.
- The emulated ROM shall be 4K Bytes
- The emulated Video RAM shall be 4K Bytes.
- System ROM shall be extracted from the original Kaypro II 2716 EPROM, converted to programmatic form, and inserted into the emulator code.

*Note: Video and System ROM actually occupy only 4K. There is a void above each of these areas. This area can be occupied by larger ROM, for example. The Kaypro II has a jumper that allows upgrading the base unit to a 2732 ROM. It has been reported that some Kaypro systems mirror System ROM (duplicate electronically). That is, System ROM repeats within the void space.

2.2.9.2 Hardware Members

Hardware Object Data Members

Member	Scope	Type	Description
cpu	Private	CPU	Object for CPU.
screen	Private	Screen	Object for Screen.
sMemory[0X10000]	Private	Short	Memory allocation.
bBank	Private	Boolean	Switch to set memory bank. True=Rom/video selected. False=RAM.
bNMIStatus	Private	Boolean	Determines the NMI line status. Initially set at false.
sNMICount	Private	Short	Determines the number of loops that NMI has remained. Initially set at 0. Must receive 20 consecutive NMI to avoid race conditions.
d[256]	Private	Device	Devices (ports) connected to the hardware. Allocated with 256 devices initally, but not defined.
bKeyAvail	Private	Boolean	Determine if a the character is available. True if character is available. Initially set as false.
sDrive	Private	Short	Determine the if the floppy drive is active. Active floppy drive. Initially set at 0.
cKeyBuff[10]	Private	Char	The Keyboard character buffer

[2] As noted in the text, bank 1 system RAM is physically the same as bank 0 system RAM. They are logically and electronically the same.

sHead	Private	Short	Head pointer for character buffer.
sTail	Private	Short	Tail pointer for character buffer.
COLUMNP	Public	Final Short	Physical line length. Maximum of length is 128. Used for Calculating the screens for formatting purposes
COLUMNS	Public	Final Short	Screen line Length. Maximum of length is 80. Used for Calculating the screens for formatting purposes
ROWS	Public	Final Short	Number of rows on screen. Maximum length of 24.
cVideo[ROWS][COLUMNP]	Global	Char	Video Memory. 24 X 128
cVideoX[ROWS][COLUMNP]	Global	Char	Video memory, translated into printable. 24 X 128

2.2.9.3 Hardware Members

Hardware Public Member Methods Summary

Methods	Scope	Parameters	Return Value	Description
Constructor		Null	Null	
Start	Public	CPU c Screen s	Null	Get references to the screen and cpu objects.
Run	Public	Null	Null	Suspends.
setBank	Public	Boolean b	None	Switches the RAM or ROM bank. True = Rom/Video bank. False = RAM.
Adddev	Public	Device din Short sPort	None	Add a device to a port.
In	Public	Short sPortIn	Short If not null Return nothing	Handles CPU IN commands. Returns the a port or a 0.
Out	Public	Short SportOut sDataOut	None	Handles CPU OUT commands. Passes in a port number.
Read	Public	Int iLocation	Synchronized short Actual memory RAM. Actual memory for the video. Actual memory for the ROM.	Handles a read request from memory as a single. Returns the memory location.
ReadWord	Public	int iLocation	Synchronized short read(iLocation) *256 +sTemp	Handles a read request from memory as a word. Returns the memory location.
Write	Public	int iLocation short sData	None	Handles a write request from memory as a single. Passes in a memory location and the memory data as a short(single).
WriteWord	Public	int iLocation int iData	None	Handles a write request from memory as a word. Passes in a memory location and the memory data as an int(word).
SetNMI	Public	None	None	Set Interrupt

GetNMI	Public	None	Synchronized boolean (true or false)	Returns status of NMI line. Must read 20 consecutive NMI's. This eliminates a race condition, where an NMI is actually generated too quickly. This can happen in the disk drive task.
NMIReset	Public	None	None	Reset the NMI line.
NMIAssert	Public	None	None	Assert the NMI line.
PutKey	Public	Char c	None	Inserts a key into the beginning of the input buffer.
GetKey	Public	None	Char cTemp	Get a key form the input buffer.
KeyAvail	Public	None	Boolean bKeyAvail	Set an input key to true as available.
GetDrive	Public	None	Short sDrive	Get the active drive.
SetDrive	Public	Short s	None	Set the active drive.
SetInt	Public	Int offset	None	Set Interrupt
GetInt	Public	None	Int offset	Returns status of Interrupt line. Must read 20 consecutive Interrupt. This eliminates a race condition, where an Interrupt is actually generated too quickly. This can happen in the disk drive task.
SetReset	Public	None	None	Set Reset line.
GetReset	Public	None	None	Sends a flag to reset the kaypro.
ReadROM	Public	Int iLocation	Short byte	Retrieves a memory location from ROM and returns the data within that location.
ReadVideo	Public	Int iLocation	Short byte	Retrieves a memory location from Video and returns the data within that location.
ReadRAM	Public	Int iLocation	Short byte	Retrieves a memory location from RAM and returns the data within that location.

2.2.10 SIO/PIO Design

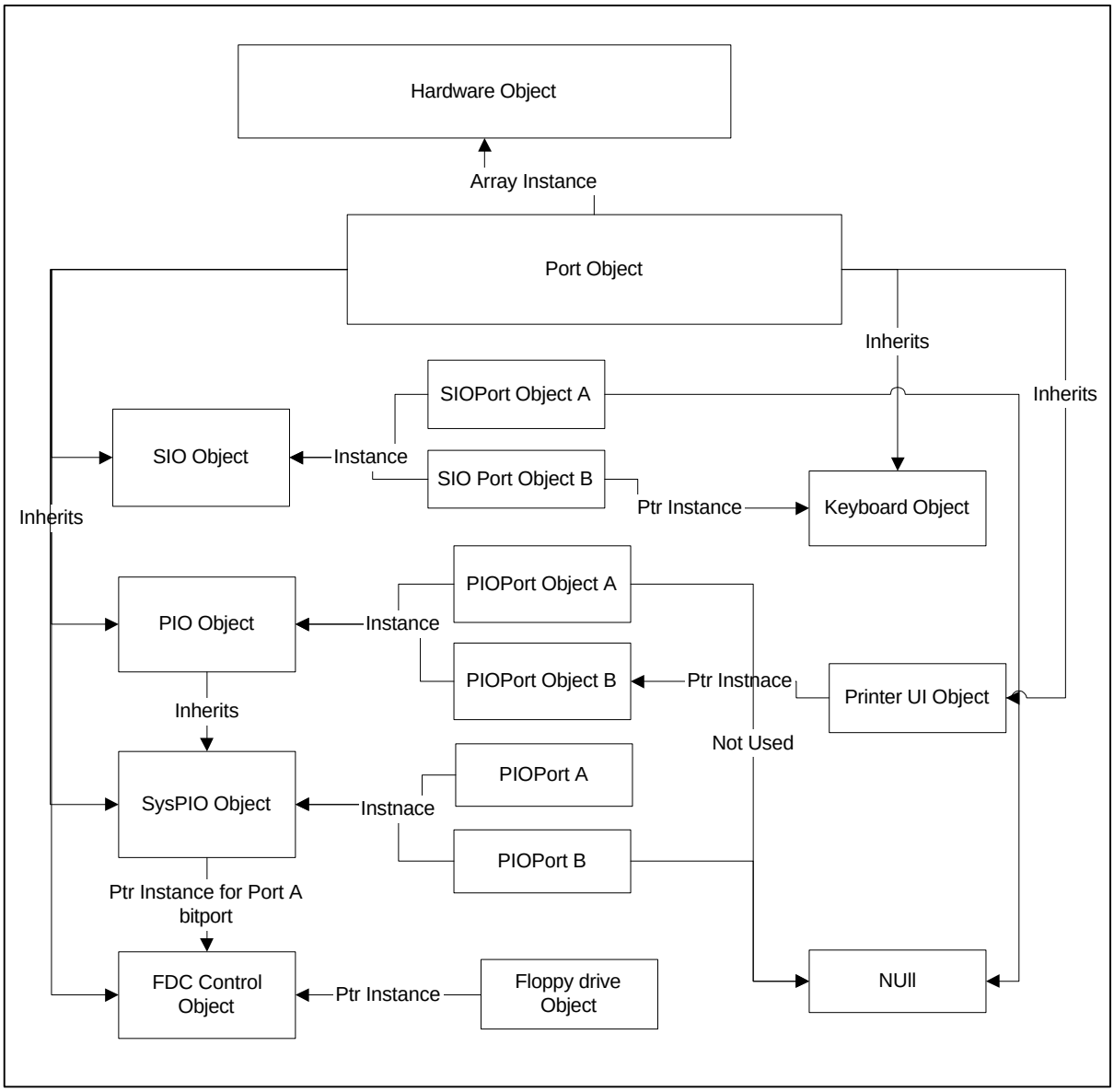


Figure 9, SIO/PIO Model

2.2.10.1 Object Method Description tables

SIO Object Method description table

Method	Description	Parameter Values	Return Value
Write	Writes the given bit pattern to the given instance of SIOPort A or B using the hex value to determine whether it's control or data. Calls SIOPort::Write passing the byte to it as well as CONTROL /DATA. SIOPort A	Port Values: 0x05 for Write data to SIO B 0x07 for Write control to SIO B The following ports will be opened but have no affect. 0x04 for Write data to SIO A	No return.

	or B determines where to put the byte. Port B is used for the keyboard.	0x06 for Write control to SIO A	
Read	Returns the value from calling SIOPort::Read (C/D) Port B is used for the keyboard.	0x05 for Read data from SIO B 0x07 for Read control from SIO B. The following ports will be opened but have no affect. 0x04 for data from SIO A 0x06 for data from SIO A	Returns short, Contents from SIOPort See (SIOPort methods)
SetHardware	Sets pointer to hardware object. For Int useage.	Pointer to hardware object.	null
GetSIO	Returns a this ptr to this SIO Object	null	Returns a SIO Object
Reset	Calls SIOPorts A and B::Reset in order to reset the SIO chip.	none	null
Constructor	Calls SIOA.Reset and SIOB.Reset Instantiates Keyboard object and passes an SIOPortB object ptr to Keyboard Calling Keyboard.SetPortObject(SIOPort B)	null	null

2.2.10.2 SIOPort Object Method Description table

Method	Description	Parameters	Return Value
Reset	WR0 Points to itself WR1 is cleared and bit 3 is set. WR2 bit 7 cleared RD0 bit 0 is cleared RD0 bit 2 is set Buffer is cleared. WR2 is copied into RD2	None	null

	Reset causes the first byte written to this port to be written into WR0, WR0 is also set to point to itself.		
Write	Uses WR0 to determine which register receives the given bit pattern if 2nd param is control. Writes byte to it internal buffer if 2nd param is data. (This will be done by keyboard)	**short** byte Value to be written **short** port States Control or Data If Port is data, Transmit buffer char avail is set (See RD1) and buffer empty is reset (See RD1). and char is written to buffer, buffer size increases. If Port is control, Byte is written to the Register pointed to by WR0	null
Read	Uses WR0 to determine which RDx register is returned if Param is CONTROL. Returns the next byte form buffer if param is DATA and decreases buffer size.	short byte that determines whether control or data is being requested.	Returns contents of RDx pointed to by WR0 if param is CONTROL. Or next byte from buffer if param is DATA.
Constructor	Construction Sequence: 1) Reset is Called.	null	null
SetPortID	Sets the string value of what port this instance of PIOPort is for Later	String PortID 'A' or 'B'	null
GetPortID	Return what port this is	null	Returns 'A' or 'B' essentialy returning MyPortID member.
SetPortObject	Connects a port object up to this port. Writes and reads for data from this port are redirected to the Connect PortObject	Object ptr that Inherits from Port	null

2.2.11 Keyboard Object Method Description table

Function	Description	Parameters	Return Value
GetKeys	Constantly polls the keyboard for input keys and places them into		

	SIOPort B's		
SetPortObject	Connects the keyboard to the given port Object.	Port inherited object	**null**

2.2.12 SIO Object Member description tables

SIOPort Members

Member	**Scope**	**Description**	**Type**
CONTROL	Public	CONTROL is a const value that is written to an SIOPort object from SIO chip to tell SIOPort that the byte passed in is a control byte.	Short
DATA	Public	DATA is a const value that is written to an SIOPort object from SIO chip object to tell SIOPort that the byte passed in is a data byte.	Short
Hardware Pointer	Private	Pointer to hardware for callback needs	Hardware Object
Buffer	Private	Holds upto 255 typed transmitted characters from keyboard. when buffer is empty sets bit 2 of RD0. (See RD0 Register)	String Array

2.2.12.1 WR0 Register Description table

Description - Pointer register and command register.

Bit value meanings
bits 0 - 2 form a pointer to the read or write register to receive the incomming byte.
bits 3 - 5 form command bits for WR0 command bits my not need be implemented as they are used in serial I/O not Keyboard input.
bits 6 - 7 CRC bits

2.2.12.2 WR1 Register Description table

Description - Contains control bits for the various interrupt and Wait/Ready modes.
This register will act as a ghost taking values that will be neglected.

Bit value meanings
bit 2 status affects vector. If bit is 0, WR2 is returned from WR2 in an Int Acknowledge sequence.
bit 3 - Interrupt mode 0
bit 4 - Interrupt mode 1

2.2.12.3 WR2 Register Description table

Description - Interrupt vector register.

Bit Value meanings
bits 0 - 7 are the ISR vector
bits 4 - 7 and 0 are always returned exactly as written
bits 1 - 3 are returned as written if bit 2 in WR1 is 0.

2.2.12.4 WR3 Register Description table

Description - Receiver logic/ control bits/ parameters
May not need to be implemented.

2.2.12.5 WR4 Register Description table

Description - Receiver and Transmitter control
May not need to be implemented.

2.2.12.6 WR5 Register Description table

Description - Transmitter control bits.
May not need to be implemented

2.2.12.7 WR6 Register Description table

Description - SDLC character for sync mode.
May not need to be implemented

2.2.12.8 WR7 Register Description table

Description - SDLC character for sync mode
May not need to be implemented

2.2.12.9 RD0 Register Description table

Description - Contains status of receive and transmit buffers

Bit Value Meanings
bit - 0 is set when a new character is available.
bit - 2 is set when buffer is empty.

2.2.12.10 RD1 Register Description table

Description - Special receive conditions/ status bits / resudue codes
May not need to be implmented

Description - ISR vector set in WR2 for non status affects..
May not need to be implemented using WR2.

2.2.13 SIO Object Members

Member	Scope	Description	Type
SIOPort A	Private	SIOPort A instantiation, controls data and control signals for the serial input and output port A.	SIOPort
SIOPort B	Private	- SIOPort B controls data and control signals for the serial input and output port B.	SIOPort
Keyboard	Private	This is an instance of the keyboard object that bill be attachted to SIO Port B	Keyboard

2.2.14 Keyboard Object Members

Member	Scope	Description	Type
SIOPort Pointer	Private	This is a ptr to SIOPort B. This port's write functions are called by keybard to place characters into SIOPort B's buffer.	SIOPort

SIO Detailed functionality notes:

-- **Hardware Connectivity** The hardware object connects to the SIO chip by instantiating it within an array of a base object called Port. The Hardware object will determine if the SIO write or read is within the base address of the SIO and just pass the port value and the byte to be written.

Example

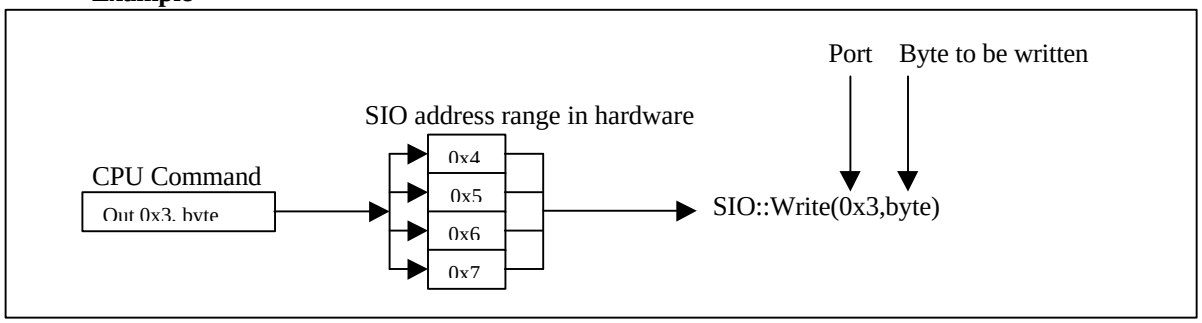


Figure 10, Hardware Port Connectivity

hardware must determine whether the port is in the SIO base address range and can call any of the Write() functions for the ports but must pass the port number and the byte to be written to the port.

Keyboard Detailed Functionality Notes

-- **Keyboard** SIO Port B is hooked up to the keyboard by passing an instance ptr of SIOPort
to keyboard. The keyboard then calls the port objects Write functions to fill the buffer with keys pressed.
Keyboard input is placed into the 255 character array until it is full.
when new char is available see Register RD0. When buffer is empty see RD0.

- Initialization
    SIO Instantiates the Keyboard Object.
    SIO calls Keyboard.SetPortObject(SIOPort B) connecting the keyboard to Port B.
    Keyboard calls GetKeys which constantly reads keys while buffer is not full.

- Example exectuion from system.
    SIO.Reset() -> SIOPortsB.Reset(), SIOPortA.Reset();
    SIO.Write(char) <- byte written into SIO WR0. WR0 pts to itself

    Assuming user programs WR0 to point to RD0.
    Keyboard.ReadKeys() Enters char.
    so SIOPort B buffer has something in it.

    Hardware calls SIO.Read(0x07) <- Port is control B so Return Port B RD0

    User program checks to see if key in buffer by testing bit 0 of RD0.

    Char is available so user calls
    SIO.Read(0x05) <- Port is data B so Return next byte from buffer.
    Buffer is empty now so SIO Port B RD0 bit 0 is cleared bit 2 is set.

- Example exectuion of keyboard
    Keyboard object polls keyboard.
    If character from keyboard, keyboard calls its ptr to SIOB.Write(char, 0x05)
    SIOB adds key to buffer and incs buffer size. Then sets bit 0 of RD0 and clears bit 2 or RD0

2.2.15 PIO Design

2.2.15.1 Object Method Description table

PIO Method Description table

Function	Description	Parameters	Return Value
Write	Writes a byte to the given Port A or B of PIO. PIO port A will be used for printer output Port B is not used. Calls PIOPort A/B.Write Passing Control or data and the byte.	short port 0x09 PIOA control 0x08 PIOA data 0x0B PIOB control 0x0A PIOB data	None
Read	Reads a byte from the given port A or B. calling PIOPort::A/B.Read (C/D)	short Port 0x09 PIOA control 0x08 PIOA data 0x0B PIOB control 0x0A PIOB data	
Constructor	Instantitaes PIOPort A and B.	null	null
GetPIO	Returns a ptr to this PIO object	null	Returns a this ptr to PIO object for hardware use.
SetPortObject	Sets the data port of the given port A or B to the passed in object that derives from Port. This will be called by Hardware or UI to connect the Print Screen to the PIOPort	**String** Port 'A' or 'B' **Port inherited object.** Object to connect to the given port data stream. that inherits from Port.	null

2.2.15.2 PIO Port Method Description table

Function	Description	Parametrs	Return Value
Read	Reads from this port. If a read is Data, if this object has a Port Object hooked up it reads from that object.	**short** C/D Parameter can be control or data	If Parameter is CONTROL then returns the contents of bitport If parameter is DATA returns null.
Write	Writes to this port. If this object has a Port Object Hooked up to it it writes to that object.	**Short** C/D **Short** byte If CONTROL writes the given byte to bitport. If DATA writes the given byte to the connected port object if a printer is connected. If connected Port Object ptr is null ie. data port is null does nothing. (No Printer connected for our use)	null
(SYSPIO only) SetPrinterReady	Sets the printerready bit of bitport (See bitport	null	null
(SYSPIO only) SetStrobe	Sets the printer stobe bit of bitport (See bitport member)	null	null
GetBit	Returns the bit value of the requested bit from bitport register	**short** bit valid ranges are 7 -0	**boolean** value of requested bit
SetBit	Sets the requested bit in the bitport register.	**short** bit valid ranges are 7 - 0	**void**
SetPortObject	Sets the PIOPort object ptr to the passed in object and all reads or writes to and from data are sent to this objects .Write(), .Read()	Object that inherits from Port.	**void**

2.2.15.3 SysPIO Method Description table

Function	Description	Parameters	Return Value
Constructor	SYSPIO Clears out both ports bitport registers and. For Both SysPIO PIOPorts A and B -> Sets bit 0 of bitport Sets bit 3 of bitport Sets bit 5 of bitport Sets bit 7 of bitport	null	null
Read	PIO A is the system bitport 0x1C is port A data 0x1D is port A control 0x1E is port B data 0x1F is port B control	**short** port If port is A or B data does nothing If Port is A or B control returns the contents of bitport by calling PIOPort A/B.Read(CONTROL).	Returns PIOPortA/B.Read (DATA) if port is data for A or B. Returns PIOPortA/B.Read(CONTROL) if port is control for A or B.
Write	PIO A is sytem bitport 0x1C is port A data 0x1D is port A control 0x1E is port B data 0x1F is port B control	**short** port **short** byte If Port is A control then writes byte to bitport register Calls PIOPortA.Write(CONTROL,byte) If Port is B control then writes the byte to B's bitport register. (WHich does nothing) Calls PIOPortB.Write(CONTROL,byte) If Port is data a or b then does nothing. Calls PIOPort::A/B.WRite(DATA,byte)	null
GetBank	Returns Memory bank conditions	Null	Returns value of system bitport used to set the memory bank Calls SysPIO::A.GetBit(7) (See Bitport member)
GetSysPIO	Returns a ptr to this SYSPIo object	null	Returns Ptr to pio object.

2.2.15.4 Object Member Description tables

PIO Member Description tables

Member	Scope	Description	Type
PIOPort A	Private	- Contains functionality of port A Connects to printer and is used for sending characters to printer screen. See PIOPort A for how PrinterScreen is connected.	PIOPort
PIOPort B	Private	Port is not Used	PIOPort
KayproII instance ptr	Public	Instance of KayproII that has the PrinterPort Function (See Printer)	KayProII

2.2.15.5 PIOPort Member Description table

Member	Scope	Description	Type
Bitport	Private	Contains status of printer, printer stobe floppy disk selection and memory bank selection for SysPIO Port A only (Port B will have similar Functionality but won't be used by the system).	Short

2.2.15.5.1 Bitport member bit patterns

Bit Value Meanings
bit 0 is Disk drive A select
bit 1 is Disk drive B select
bit 2 not used
bit 3 is printer ready flag
bit 4 is centronics data strobe
bit 5 double density select
bit 6 disk drive motor on
bit 7 memory bank select (See Hung for values)

Port Object	Inherited Object	An object that inherits from Port and has a Read and Write function Reads and writes for data are redirected to this object if it is not null. The UI or Hardware will call PIO::SetPortObject('A', PrinterScreen) to connect the printer to this port.	Port

-***PIOPort Functionality Description***

PIOPort will have a Port Object that is instantiaed to null. Writes to data of PIOPort are sent to its Object.

-- **PIOPort Write Example**

-Initialization

Hardware calls PIO.SetPortObject('A',PrinterScreen)

PIO calls PIOPort A.SetPortObject(PrinterScreen)

-Sample printing

Hardware calls PIO.Write(0x08,'a')

PIO Calls PIOPortA.Write('a');

PIOPortA calls it's ptr instance to the connected object s.Write('a') (this will be the printer UI screen)

the printer UI Screen receives the given character.

2.2.15.6 FDC Floppy Disk Controller Methods

Method	Scope	Parameter values	Return Values	Description
SetSysPIO	Public	PIOPort Object	Null	Sets a pointer of the SystemPioPort A for checking working drive letter

2.2.15.7 FDC Floppy Disk Controller Members

Member	Scope	Description	Type
PIOPort pointer	Private	A Pointer instance to the SysPIOPort A. Used in accessing which Drive is the FDC working with	PIOPort
DataRegister	Private	Holds Data that is either read from the floppy dirve or to be the floppy drive. Port 0x13	Short
TrackRegister	Private	Holds track number of the Current Read or write position in the floppy drive. Port 0x11	Short
SectorRegister	Private	Holds the address of the desired read or write sector. Port 0x12	Short
CommandRegister	Private	Write only register that holds the current command Port 0x10 Bit Patterns are	Short
StatusRegister	Private	Holds status bits of the disk Drive (busy bit used only)	Short

2.2.15.7.1 Status Register Bits

Bit	Description
0	Busy Bit. Is set when reading Cleared when not.

2.2.15.7.2 Command Register Bit Patterns

Command	Bit Pattern
Read Sector	100
Write Sector	001

2.2.15.8 Related Members from other objects

Object	Method	Description	Params	ReturnValues
Hardware	Int	Interrupt method called by SIO	short Byte, contents of WR2 or RD2 which is a half address ISR Vector.	Iei Possibly
UI	GetPrinterScreen	Returns a ptr object to the printer screen	void	UI::PrinterScreen object ptr.
Printer	PdataOut(byte)	Kaypro::Printer screen write function	short byte	Null

2.2.16 Bootstrap Loader

The Kaypro II utilizes a unique way of loading the CP/M operating system. Older systems required manually loading the bootstrap code.

The Kaypro II. Uses an internal ROM. This ROM contains the startup code needed to bring CP/M into memory.

Typical CP/M diskettes contained a short bootstrap program on the lowest track and sector. The Kaypro II stores the location to load CP/M and length of the CP/M operating system in this area instead. This should be noted. This should not be a problem for the emulator if the CP/M floppy diskettes are faithfully duplicated.

2.2.17 Operating system

The operating system shall be supported on disk images. The disk images shall contain CP/M 2.2. The operating system shall be read from a valid Kaypro II diskette, and transferred electronically into a form recognizable by the Kaypro II emulator. Once inside the emulator, the disk images shall be loaded via one of two virtual floppy disk drives.

- CP/M 2.2 shall be supported
- CP/M OS shall be supplied on track 1 of each virtual floppy disk.
- The emulator shall support loading of the OS from floppy drive A
- The CP/M operating system shall actually be run at the software level via an obtained copy of the CP/M operating system

3. Project Deliverables

This section identifies all deliverable components of the project including hardware, software, training, and documentation.

3.1 Hardware

No hardware shall be delivered

3.2 Software

All Kaypro II software shall be delivered.
All source code shall be delivered.
All associated build or make files shall be delivered

3.3 Training

No training shall be provided.

3.4 Project Documentation

There are two categories of documents: project development documents, such as the project plan and design specification, and customer documents, such as the user's guide. These documents are delivered according to the project schedule.

3.4.1 Project Development Documentation

Requirements design documents shall be provided
Requirements specifications document shall be provided
Design documents shall be provided

3.4.2 Customer/Operations Documentation

A user guide shall be provided

4. Applicable Documents, Reference, and Glossary

This section contains title, author, and publication information for documents referred to or having an impact on the requirements for this project. It also contains a comprehensive glossary of applicable terms and acronyms.

4.1 References

Requirements Definition For The CSI426/Kaypro II Emulator
Zilog Z80 Microprocessor Family User's Manual, Part number Q1/95 DC 8309-1
*Z80.DOC, opcode reference, compiled by Sean Young (*syoung@cs.vu.nl*)*
Synertek Data Book, 1983

4.2 Appendix A, Z-80 Opcodes

4.2.1 8 bit Load Group

Mnemonic	Symbolic Operation	S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD r, r'	r ← r'	•	•	•	•	•	•	•	•	01 r r'		1	1	4	r, r' Reg.
LD p, p'*	p ← p'	•	•	•	•	•	•	•	•	11 011 101 01 p p'	DD	2	2	8	000 B 001 C
LD q, q'*	q ← q'	•	•	•	•	•	•	•	•	11 111 101 01 q q'	FD	2	2	8	010 D 011 E
LD r, n	r ← n	•	•	•	•	•	•	•	•	00 r 110 ← n →		2	2	7	100 H 101 L
LD p, n*	p ← n	•	•	•	•	•	•	•	•	11 011 101 00 p 110 ← n →	DD	3	3	11	111 A p, p' Reg.
LD q, n*	q ← n	•	•	•	•	•	•	•	•	11 111 101 00 q 110 ← n →	FD	3	3	11	000 B 001 C 010 D
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	•	•	01 r 110		1	2	7	011 E
LD r, (IX + d)	r ← (IX + d)	•	•	•	•	•	•	•	•	11 011 101 01 r 110 ← d →	DD	3	5	19	100 IX_H 101 IX_L 111 A
LD r, (IY + d)	r ← (IY + d)	•	•	•	•	•	•	•	•	11 111 101 01 r 110 ← d →	FD	3	5	19	q, q' Reg. 000 B
LD (HL), r	(HL) ← r	•	•	•	•	•	•	•	•	01 110 r		1	2	7	001 C
LD (IX + d), r	(IX + d) ← r	•	•	•	•	•	•	•	•	11 011 101 01 110 r ← d →	DD	3	5	19	010 D 011 E 100 IY_H
LD (IY + d), r	(IY + d) ← r	•	•	•	•	•	•	•	•	11 111 101 01 110 r ← d →	FD	3	5	19	101 IY_L 111 A
LD (HL), n	(HL) ← n	•	•	•	•	•	•	•	•	00 110 110 ← n →	36	2	3	10	
LD (IX + d), n	(IX + d) ← n	•	•	•	•	•	•	•	•	11 011 101 00 110 110 ← d → ← n →	DD 36	4	5	19	
LD (IY + d), n	(IY + d) ← n	•	•	•	•	•	•	•	•	11 111 101 00 110 110 ← d → ← n →	FD 36	4	5	19	
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	•	•	00 111 010 ← n → ← n →	3A	3	4	13	
LD (BC), A	(BC) ← A	•	•	•	•	•	•	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	•	•	•	•	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	•	•	•	•	•	•	00 110 010 ← n → ← n →	32	3	4	13	
LD A, I	A ← I	↕	↕	↕	0	↕	IFF_2	0	•	11 101 101 01 010 111	ED 57	2	2	9	
LD A, R	A ← R	↕	↕	↕	0	↕	IFF_2	0	•	11 101 101 01 011 111	ED 5F	2	2	9	R is read after it is increased.
LD I, A	I ← A	•	•	•	•	•	•	•	•	11 101 101 01 000 111	ED 47	2	2	9	
LD R, A	R ← A	•	•	•	•	•	•	•	•	11 101 101 01 001 111	ED 4F	2	2	9	R is written after i is increased.

Notes:
r, r' means any of the registers A, B, C, D, E, H, L.
p, p' means any of the registers A, B, C, D, E, IX_H, IX_L.
q, q' means any of the registers A, B, C, D, E, IY_H, IY_L.
dd_L, dd_H refer to high order and low order eight bits of the register respectively.

* means unofficial instruction.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set,
↕ = flag is set according to the result of the operation, IFF_2 = the interrupt flip-flop 2 is copied.

4.2.2 16 bit Load Group

Mnemonic	Symbolic Operation	S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	•	•	•	•	•	•	00 dd0 001 ← n → ← n →		3	3	10	dd Pair 00 BC 01 DE
LD IX, nn	IX ← nn	•	•	•	•	•	•	•	•	11 011 101 00 110 001 ← n → ← n →	DD 21	4	4	14	02 HL 03 SP
LD IY, nn	IY ← nn	•	•	•	•	•	•	•	•	11 111 101 00 110 001 ← n → ← n →	FD 21	4	4	14	
LD HL, (nn)	L ← (nn) H ← (nn+1)	•	•	•	•	•	•	•	•	00 101 010 ← n → ← n →	2A	3	5	16	
LD dd, (nn)	dd_L ← (nn) dd_H ← (nn+1)	•	•	•	•	•	•	•	•	11 101 101 01 dd1 011 ← n → ← n →	ED	4	6	20	
LD IX, (nn)	IX_L ← (nn) IX_H ← (nn+1)	•	•	•	•	•	•	•	•	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY, (nn)	IY_L ← (nn) IY_H ← (nn+1)	•	•	•	•	•	•	•	•	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn) ← L (nn+1) ← H	•	•	•	•	•	•	•	•	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn) ← dd_L (nn+1) ← dd_H	•	•	•	•	•	•	•	•	11 101 101 01 dd0 011 ← n → ← n →	DD	4	6	20	
LD (nn), IX	(nn) ← IX_L (nn+1) ← IX_H	•	•	•	•	•	•	•	•	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn) ← IY_L (nn+1) ← IY_H	•	•	•	•	•	•	•	•	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP ← HL	•	•	•	•	•	•	•	•	11 111 001	F9	1	1	6	
LD SP, IX	SP ← IX	•	•	•	•	•	•	•	•	11 011 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP ← IY	•	•	•	•	•	•	•	•	11 111 101 11 111 001	FD F9	2	2	10	
PUSH qq	SP ← SP - 1 (SP) ← qq_H SP ← SP - 1 (SP) ← qq_L	•	•	•	•	•	•	•	•	11 qq0 101		1	3	11	qq Pair 00 BC 01 DE 10 HL
PUSH IX	SP ← SP - 1 (SP) ← IX_H SP ← SP - 1 (SP) ← IX_L	•	•	•	•	•	•	•	•	11 011 101 11 100 101	DD E5	2	4	15	11 AF
PUSH IY	SP ← SP - 1 (SP) ← IY_H SP ← SP - 1 (SP) ← IY_L	•	•	•	•	•	•	•	•	11 111 101 11 100 101	FD E5	2	4	15	

POP qq	(SP) ← qq_L SP ← SP + 1 (SP) ← qq_H SP ← SP + 1	•	•	•	•	•	•	•	•	11 qq0 001		1	3	10
POP IX	(SP) ← IX_L SP ← SP + 1 (SP) ← IX_H SP ← SP + 1	•	•	•	•	•	•	•	•	11 011 101 11 100 001	DD E1	2	4	14
POP IY	(SP) ← IY_L SP ← SP + 1 (SP) ← IY_H SP ← SP + 1	•	•	•	•	•	•	•	•	11 111 101 11 100 001	FD E1	2	4	14

Notes: dd is any of the register pair BC, DE, HL, SP.
qq is any of the register pair BC, DE, HL, AF.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.3 Exchange, Block Transfer and Search Groups

Mnemonic	Symbolic Operation	S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	•	•	11 101 011	EB	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	•	•	00 001 000	08	1	1	4	
EXX	BC ↔ BC' DE ↔ DE' HL ↔ HL'	•	•	•	•	•	•	•	•	11 011 001	D9	1	1	4	
EX (SP), HL	(SP+1) ↔ H (SP) ↔ L	•	•	•	•	•	•	•	•	11 100 011	E3	1	5	19	
EX (SP), IX	(SP+1) ↔ IX_H (SP) ↔ IX_L	•	•	•	•	•	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	(SP+1) ↔ IY_H (SP) ↔ IY_L	•	•	•	•	•	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC - 1	•	•	↕[1]	0	↕[2]	↕[3]	0	•	11 101 101 10 100 000	ED A0	2	4	16	
LDIR	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC - 1 repeat until: BC = 0	•	•	↕[1]	0	↕[2]	0	0	•	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	if BC ≠ 0 if BC = 0
LDD	(DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1	•	•	↕[1]	0	↕[2]	↕[3]	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1 repeat until: BC = 0	•	•	↕[1]	0	↕[2]	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	if BC ≠ 0 if BC = 0
CPI	A - (HL) HL ← HL + 1 BC ← BC -1	↕[4]	↕[4]	↕[5]	↕[4]	↕[6]	↕[3]	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A - (HL) HL ← HL + 1 BC ← BC -1 Repeat until: A = (HL) or BC = 0	↕[4]	↕[4]	↕[5]	↕[4]	↕[6]	↕[3]	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	if BC ≠ 0 and A ≠ (HL). if BC = 0 or A = (HL)
CPD	A - (HL) HL ← HL - 1 BC ← BC -1	↕[4]	↕[4]	↕[5]	↕[4]	↕[6]	↕[3]	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A - (HL) HL ← HL - 1 BC ← BC -1 Repeat until: A = (HL) or BC = 0	↕[4]	↕[4]	↕[5]	↕[4]	↕[6]	↕[3]	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	if BC ≠ 0 and A ≠ (HL). if BC = 0 or A = (HL)

Notes:
[1] F5 is a copy of bit 1 of A + last transferred byte, thus $(A + (HL))_1$
[2] F3 is a copy of bit 3 of A + last transferred byte, thus $(A + (HL))_3$
[3] P/V flag is 0 if the result of BC - 1 = 0, otherwise P/V = 1.
[4] These flags are set as in CP (HL)
[5] F5 is copy of bit 1 of A - last compared address - H, thus $(A - (HL) - H)_1$. H is as in F after the comparison.
[6] F3 is copy of bit 3 of A - last compared address - H, thus $(A - (HL) - H)_3$. H is as in F after the comparison.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.4 8 bit Arithmetic and Logical Group

Mnemonic	Symbolic Operation	Flags S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
ADD A, r	A ← A + r	↕	↕	↕	↕	↕	V	0	↕	10 000 r		1	1	4	r Reg. p Re
ADD A, p*	A ← A + p	↕	↕	↕	↕	↕	V	0	↕	11 011 101 10 000 p	DD	2	2	8	000 B 000 B 001 C 001 C
ADD A, q*	A ← A + q	↕	↕	↕	↕	↕	V	0	↕	11 111 101 10 000 q	FD	2	2	8	010 D 010 D 011 E 011 E
ADD A, n	A ← A + n	↕	↕	↕	↕	↕	V	0	↕	11 000 110 ← n →		2	2	8	100 H 100 IX 101 L 101 IX
ADD A, (HL)	A ← A + (HL)	↕	↕	↕	↕	↕	V	0	↕	10 000 110		1	2	7	111 A 111 A
ADD A, (IX + d)	A ← A + (IX + d)	↕	↕	↕	↕	↕	V	0	↕	11 011 101 10 000 110 ← d →	DD	3	5	19	
ADD A, (IY + d)	A ← A + (IY + d)	↕	↕	↕	↕	↕	V	0	↕	11 111 101 10 000 110 ← d →	FD	3	5	19	
ADC A, s	A ← A + s + CY	↕	↕	↕	↕	↕	V	0	↕	001					s is any of r, n, (HL)
SUB A, s	A ← A - s	↕	↕	↕	↕	↕	V	1	↕	010					(IX+d), (IY+d), p, q
SBC A, s	A ← A - s - CY	↕	↕	↕	↕	↕	V	1	↕	011					as shown for the AD
AND s	A ← A AND s	↕	↕	↕	1	↕	P	0	0	100					instruction. The
OR s	A ← A OR s	↕	↕	↕	0	↕	P	0	0	110					underlined bits replace
XOR s	A ← A XOR s	↕	↕	↕	0	↕	P	0	0	101					the underlined bits i
CP s	A - s	↕	↕	↕[1]	↕	↕[1]	V	1	↕	111					the ADD set.
INC r	r ← r + 1	↕	↕	↕	↕	↕	V	0	•	00 r 100		1	1	4	
INC p*	p ← p + 1	↕	↕	↕	↕	↕	V	0	•	11 011 101 00 p 100	DD	2	2	8	q Reg. 000 B
INC q*	q ← q + 1	↕	↕	↕	↕	↕	V	0	•	11 111 101 00 q 100	FD	2	2	8	001 C 010 D
INC (HL)	(HL) ← (HL) + 1	↕	↕	↕	↕	↕	V	0	•	00 110 100		1	3	11	011 E
INC (IX + d)	(IX + d) ← (IX + d) + 1	↕	↕	↕	↕	↕	V	0	•	11 011 101 00 110 100 ← d →	DD	3	6	23	100 IY_H 101 IY_L 111 A
INC (IY + d)	(IY + d) ← (IY + d) + 1	↕	↕	↕	↕	↕	V	0	•	11 111 101 00 110 100 ← d →	FD	3	6	23	
DEC m	m ← m - 1	↕	↕	↕	↕	↕	V	1	•	101					m is any of r, p, q, (HL), (IX+d), (IY+d), as shown for the IN instruction. DEC same format and states as INC. Replace 100 with 1 in opcode.

Notes:
[1] F5 and F3 are copied from the operand (s), not from the result of (A - s).
The V symbol in the P/V flag column indicates that the P/V flags contains the overflow of the operation. Similarly the P symbol indicates parity.
r means any of the registers A, B, C, D, E, H, L.
p means any of the registers A, B, C, D, E, IX_H, IX_L.
q means any of the registers A, B, C, D, E, IY_H, IY_L.
dd_L, dd_H refer to high order and low order eight bits of the register respectively.
CY means the carry flip-flop.
* means unofficial instruction.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.5 16 bit Arithmetic Group

Mnemonic	Symbolic Operation	Flags S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
ADD HL, ss	HL ← HL + ss	•	•	↕[2]	↕[2]	↕[2]	•	0	↕[1]	00 ss1 001		1	3	11	ss Reg.
ADC HL, ss	HL ← HL + ss + CY	↕[1]	↕[1]	↕[2]	↕[2]	↕[2]	V[1]	0	↕[1]	11 101 101 01 ss1 010	ED	2	4	15	00 BC 01 DE
SBC HL, ss	HL ← HL - ss - CY	↕[1]	↕[1]	↕[2]	↕[2]	↕[2]	V[1]	1	↕[1]	11 101 101 01 ss0 010	ED	2	4	15	10 HL 11 SP
ADD IX, pp	IX ← IX + pp	•	•	↕[2]	↕[2]	↕[2]	•	0	↕[1]	11 011 101 00 pp1 001	DD	2	4	15	pp Reg.
ADD IY, rr	IY ← IY + rr	•	•	↕[2]	↕[2]	↕[2]	•	0	↕[1]	11 111 101 00 rr1 001	FD	2	4	15	00 BC 01 DE
INC ss	ss ← ss + 1	•	•	•	•	•	•	•	•	00 ss0 011		1	1	6	10 IX
INC IX	IX ← IX + 1	•	•	•	•	•	•	•	•	11 011 101 00 100 011	DD 23	2	2	10	11 SP
INC IY	IY ← IY + 1	•	•	•	•	•	•	•	•	11 111 101 00 100 011	FD 23	2	2	10	rr Reg. 00 BC
DEC ss	ss ← ss - 1	•	•	•	•	•	•	•	•	00 ss1 011		1	1	6	01 DE
DEC IX	IX ← IX - 1	•	•	•	•	•	•	•	•	11 011 101 00 101 011	DD 2B	2	2	10	10 IY 11 SP
DEC IY	IY ← IY - 1	•	•	•	•	•	•	•	•	11 111 101 00 101 011	FD 2B	2	2	10	

Notes:
The V symbol in the P/V flag column indicates that the P/V flags contains the overflow of the operation.
ss means any of the registers BC, DE, HL, SP.
pp means any of the registers BC, DE, IX, SP.
rr means any of the registers BC, DE, IY, SP.
16 bit additions are performed by first adding the two low order eight bits, and then the two high order eight bits.
[1] Indicates the flag is affected by the 16 bit result of the operation.
[2] Indicates the flag is affected by the 8 bit addition of the high order eight bits.
CY means the carry flip-flop.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.6 General Purpose Arithmetic and CPU Control Groups

Mnemonic	Symbolic Operation	Flags S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
DAA	Converts A into packed BCD following add or subtract with BCD operands.	↕	↕	↕	↕	↕	P	•	↕	00 100 111	27	1	1	4	
CPL	$A \leftarrow \overline{A}$	•	•	↕[1]	1	↕[1]	•	1	•	00 101 111	2F	1	1	4	One's complement.
NEG[4]	A ← 0 - A	↕	↕	↕	↕	↕	V	1	↕	11 101 101 01 000 100	ED 44	2	2	8	Two's complement.
CCF	$CY \leftarrow \overline{CY}$	•	•	↕[1]	↕[2]	↕[1]	•	0	↕	00 111 111	3F	1	1	4	Complement carry flag.
SCF	CY ← 1	•	•	↕[1]	0	↕[1]	•	0	1	00 110 111	37	1	1	4	
NOP	No operations	•	•	•	•	•	•	•	•	00 000 000	00	1	1	4	
HALT	CPU halted	•	•	•	•	•	•	•	•	01 110 110	76	1	1	4	
DI[3]	$IFF_1 \leftarrow 0$ $IFF_2 \leftarrow 0$	•	•	•	•	•	•	•	•	11 110 011	F3	1	1	4	
EI[3]	$IFF_1 \leftarrow 1$ $IFF_2 \leftarrow 1$	•	•	•	•	•	•	•	•	11 111 011	FB	1	1	4	
IM 0[4]	Set interrupt mode 0	•	•	•	•	•	•	•	•	11 101 101 01 000 110	ED 46	2	2	8	
IM 1[4]	Set interrupt mode 1	•	•	•	•	•	•	•	•	11 101 101 01 010 110	ED 56	2	2	8	
IM 2[4]	Set interrupt mode 2	•	•	•	•	•	•	•	•	11 101 101 01 011 110	ED 5E	2	2	8	

Notes: The V symbol in the P/V flag column indicates that the P/V flags contains the overflow of the operation. Similarly the P symbo indicates parity.

[1] F5 and F3 are a copy of bit 5 and 3 of register A

[2] H contains the previous carry state (after instruction $H \leftrightarrow C$)

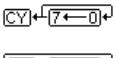
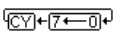
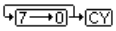
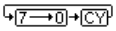
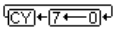
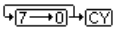
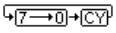
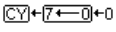
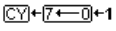
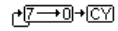
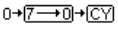
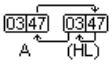
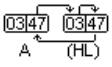
[3] No interrupts are issued directly after a DI or EI.

[4] This instruction has other unofficial opcodes, see Opcodes list.

CY means the carry flip-flop.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.7 Rotate and Shift Group

Mnemonic	Symbolic Operation	Flags S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		•	•	↕	0	↕	•	0	↕	00 000 111	07	1	1	4	
RLA		•	•	↕	0	↕	•	0	↕	00 010 111	17	1	1	4	
RRCA		•	•	↕	0	↕	•	0	↕	00 001 111	0F	1	1	4	
RRA		•	•	↕	0	↕	•	0	↕	00 011 111	1F	1	1	4	
RLC r		↕	↕	↕	0	↕	P	0	↕	11 001 011 00 000 r	CB	2	2	8	r Reg. 000 B
RLC (HL)		↕	↕	↕	0	↕	P	0	↕	11 001 011 00 000 110	CB	2	4	15	001 C 010 D
RLC (IX + d)		↕	↕	↕	0	↕	P	0	↕	11 011 101 11 001 011 ← d → 00 000 110	DD CB	4	6	23	011 E 100 H 101 L 111 A
RLC (IY + d)		↕	↕	↕	0	↕	P	0	↕	11 111 101 11 001 011 ← d → 00 000 110	FD CB	4	6	23	
LD r,RLC (IX + d)*	r ← (IX + d) RLC r (IX + d) ← r	↕	↕	↕	0	↕	P	0	↕	11 011 101 11 001 011 ← d → 00 000 r	DD CB	4	6	23	
LD r,RLC (IY + d)*	r ← (IY + d) RLC r (IY + d) ← r	↕	↕	↕	0	↕	P	0	↕	11 111 101 11 001 011 ← d → 00 000 r	FD CB	4	6	23	
RL m		↕	↕	↕	0	↕	P	0	↕	010					Instruction format and states are the same as RLC. Replace 000 with new number.
RRC m		↕	↕	↕	0	↕	P	0	↕	001					
RR m		↕	↕	↕	0	↕	P	0	↕	011					
SLA m		↕	↕	↕	0	↕	P	0	↕	100					
SLL m*		↕	↕	↕	0	↕	P	0	↕	110					
SRA m		↕	↕	↕	0	↕	P	0	↕	101					
SRL m		↕	↕	↕	0	↕	P	0	↕	111					
RLD		↕	↕	↕	0	↕	P	0	•	11 101 101 01 101 111	ED 6F	2	5	18	
RRD		↕	↕	↕	0	↕	P	0	•	11 101 101 01 100 111	ED 67	2	5	18	

Notes: The P symbol in the P/V flag column indicates that the P/V flags contains the parity of the result.
r means any of the registers A, B, C, D, E, H, L.
* means unofficial instruction.
CY means the carry flip-flop.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.8 Bit Manipulation Group

Mnemonic	Symbolic Operation	Flags S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r_b}$	↕[1]	↕	↕[2]	1	↕[3]	↕[4]	0	•	11 001 011 01 b r	CB	2	2	8	r Reg. 000 B
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	↕[1]	↕	↕[2]	1	↕[3]	↕[4]	0	•	11 001 011 01 b 110	CB	2	3	12	001 C 010 D
BIT b, (IX + d)[5]	$Z \leftarrow \overline{(IX + d)_b}$	↕[1]	↕	↕[2]	1	↕[3]	↕[4]	0	•	11 011 101 11 001 011 ← d → 01 b 110	DD CB	4	5	20	011 E 100 H 101 L 111 A
BIT b, (IY + d)[5]	$Z \leftarrow \overline{(IY + d)_b}$	↕[1]	↕	↕[2]	1	↕[3]	↕[4]	0	•	11 111 101 11 001 011 ← d → 01 b 110	FD CB	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	•	•	11 001 011 11 b r	CB	2	2	8	b Bit. 000 0 001 1
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	•	•	11 001 011 11 b 110	CB	2	4	15	010 2 011 3
SET b, (IX + d)	$(IX + d)_b \leftarrow 1$	•	•	•	•	•	•	•	•	11 011 101 11 001 011 ← d → 11 b 110	DD CB	4	6	23	100 4 101 5 110 6 111 7
SET b, (IY + d)	$(IY + d)_b \leftarrow 1$	•	•	•	•	•	•	•	•	11 111 101 11 001 011 ← d → 11 b 110	FD CB	4	6	23	
LD r,SET b, (IX + d)*	$r \leftarrow (IX + d)$ $r_b \leftarrow 1$ $(IX + d) \leftarrow r$	•	•	•	•	•	•	•	•	11 011 101 11 001 011 ← d → 11 b r	DD CB	4	6	23	
LD r,SET b, (IY + d)*	$r \leftarrow (IY + d)$ $r_b \leftarrow 1$ $(IY + d) \leftarrow r$	•	•	•	•	•	•	•	•	11 111 101 11 001 011 ← d → 11 b r	FD CB	4	6	23	
RES b, m	$m_b \leftarrow 0$ m ≡ r, (HL), (IX+d), (IY+d)	•	•	•	•	•	•	•	•	10					To form new opcode replace 11 of SET b, s with 10. Flags and states are the same.

Notes:
The notation m_b indicates bit b (0 to 7) of location m.
BIT instructions are performed by an bitwise AND.
[1] S is set if b = 7 and Z = 0
[2] F5 is set if b = 5 and Z = 0
[3] F3 is set if b = 3 and Z = 0
[4] P/V is set like the Z flag
[5] This instruction has other unofficial opcodes
* means unofficial instruction.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.9 Input and Output Groups

Mnemonic	Symbolic Operation	S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
IN A, (n)	A ← (n)	•	•	•	•	•	•	•	•	11 011 011 ← n →	DB	2	3	11	r Reg. 000 B
IN r, (C)	r ← (C)	↕	↕	↕	0	↕	P	0	•	11 101 101 01 r 000	ED	2	3	12	001 C 010 D
IN (C)* or IN F, (C)*	Just affects flags, value is lost.	↕	↕	↕	0	↕	P	0	•	11 101 101 01 110 000	ED 70	2	3	12	011 E 100 H
INI	(HL) ← (C) HL ← HL + 1 B ← B - 1	↕[1]	↕[1]	↕[1]	↕[3]	↕[1]	X	↕[2]	↕[3]	11 101 101 10 100 010	ED A2	2	4	16	101 L 111 A
INIR	(HL) ← (C) HL ← HL + 1 B ← B - 1 Repeat until B = 0	0	1	0	↕[3]	0	X	↕[2]	↕[3]	11 101 101 10 110 010	ED B2	2 2	5 4	21 16	if B ≠ 0 if B = 0
IND	(HL) ← (C) HL ← HL - 1 B ← B - 1	↕[1]	↕[1]	↕[1]	↕[4]	↕[1]	X	↕[2]	↕[4]	11 101 101 10 101 010	ED AA	2	4	16	
INDR	(HL) ← (C) HL ← HL - 1 B ← B - 1 Repeat until B = 0	0	1	0	↕[4]	0	X	↕[2]	↕[4]	11 101 101 10 111 010	ED BA	2 2	5 4	21 16	if B ≠ 0 if B = 0
OUT (n), A	(n) ← A	•	•	•	•	•	•	•	•	11 010 011 ← n →	D3	2	3	11	
OUT (C), r	(C) ← r	•	•	•	•	•	•	•	•	11 101 101 01 r 001	ED	2	3	12	
OUT (C), 0*	(C) ← 0	•	•	•	•	•	•	•	•	11 101 101 01 110 001	ED 71	2	3	12	
OUTI	(C) ← (HL) HL ← HL + 1 B ← B - 1	↕[1]	↕[1]	↕[1]	X	↕[1]	X	X	X	11 101 101 10 100 011	ED A3	2	4	16	
OTIR	(C) ← (HL) HL ← HL + 1 B ← B - 1 Repeat until B = 0	0	1	0	X	0	X	X	X	11 101 101 10 110 011	ED B3	2 2	5 4	21 16	if B ≠ 0 if B = 0
OUTD	(C) ← (HL) HL ← HL - 1 B ← B - 1	↕[1]	↕[1]	↕[1]	X	↕[1]	X	X	X	11 101 101 10 101 011	ED AB	2	4	16	
OTDR	(C) ← (HL) HL ← HL - 1 B ← B - 1 Repeat until B = 0	0	1	0	X	0	X	X	X	11 101 101 10 111 011	ED BB	2 2	5 4	21 16	if B ≠ 0 if B = 0

Notes:
The V symbol in the P/V flag column indicates that the P/V flags contains the overflow of the operation. Similarly the P symbol indicates parity.
r means any of the registers A, B, C, D, E, H, L.
[1] flag is affected by the result of B ← B - 1 as in DEC B.
[2] N is a copy bit 7 of the last value from the input (C).
[3] this flag contains the carry of (((C + 1) AND 255) + (C))
[4] this flag contains the carry of (((C - 1) AND 255) + (C))
* means unofficial instruction.

Flag Notation:
• = flag is not affected, 0 = flag is reset, 1 = flag is set, X = flag is unknown,
↕ = flag is set according to the result of the operation.

4.2.10 Jump Group

Mnemonic	Symbolic Operation	Flags S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
JP nn	PC ← nn	•	•	•	•	•	•	•	•	11 000 011 ← n → ← n →	C3	3	3	10	
JP cc, nn	if cc is true, PC ← nn	•	•	•	•	•	•	•	•	11 ccc 010 ← n → ← n →		3	3	10	ccc Condition 000 NZ 001 Z 010 NC 011 C 100 PO 101 PE 110 P 111 M
JR e	PC ← PC + e	•	•	•	•	•	•	•	•	00 011 000 ← e - 2 →	18	2	3	12	
JR ss, e	if ss is true	•	•	•	•	•	•	•	•	00 ss 000		2	3	12	if ss is true
	PC ← PC + e									← e - 2 →		2	2	7	if ss is false
JP HL	PC ← HL	•	•	•	•	•	•	•	•	11 101 001	E9	1	1	4	
JP IX	PC ← IX	•	•	•	•	•	•	•	•	11 011 101 11 101 001	DD E9	2	2	8	ss Condition 111 C 110 NC
JP IY	PC ← IY	•	•	•	•	•	•	•	•	11 111 101 11 101 001	FD E9	2	2	8	101 Z 100 NZ
DJNZ e	B ← B - 1 if B ≠ 0 PC ← PC + e	•	•	•	•	•	•	•	•	00 010 000 ← e - 2 →	10	2 2	2 3	8 13	if B = 0 if B ≠ 0

Notes: e is a signed two-complement number in the range <-126, 129>
e - 2 in the opcode provides an effective number of PC + e as PC incremented by 2 prior to the addition of e.

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.2.11 Call and Return Group

Mnemonic	Symbolic Operation	S	Z	F5	H	F3	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
CALL nn	SP ← SP - 1 (SP) ← PC_H SP ← SP - 1 (SP) ← PC_L PC ← nn	•	•	•	•	•	•	•	•	11 001 101 ← n → ← n →	CD	3	5	17	
CALL cc, nn	if cc is true, SP ← SP - 1 (SP) ← PC_H SP ← SP - 1 (SP) ← PC_L PC ← nn	•	•	•	•	•	•	•	•	11 ccc 100 ← n → ← n →		3 3 3	3 5 5	10 17 17	if cc is false if cc is true
RET	PC_L ← (SP) SP ← SP + 1 PC_H ← (SP) SP ← SP + 1	•	•	•	•	•	•	•	•	11 001 001	C9	1	3	10	
RET cc	if cc is true, PC_L ← (SP) SP ← SP + 1 PC_H ← (SP) SP ← SP + 1	•	•	•	•	•	•	•	•	11 ccc 000		1 1	1 3	5 11	if cc is false if cc is true
RETI[2]	PC_L ← (SP) SP ← SP + 1 PC_H ← (SP) SP ← SP + 1	•	•	•	•	•	•	•	•	11 101 101 01 001 101	ED 4D	2	4	14	cc Condition 000 NZ 001 Z 010 NC 011 C
RETN[1,2]	PC_L ← (SP) SP ← SP + 1 PC_H ← (SP) SP ← SP + 1 IFF_1 ← IFF_2	•	•	•	•	•	•	•	•	11 101 101 01 000 101	ED 45	2	4	14	100 PO 101 PE 110 P 111 M
RST p	SP ← SP - 1 (SP) ← PC_H SP ← SP - 1 (SP) ← PC_L PC ← p	•	•	•	•	•	•	•	•	11 t 111		1	3	11	t p 000 0h 001 8h 010 10h 011 18h 100 20h 101 28h 110 30h 111 38h

Notes: [1] This instruction has other unofficial opcodes, see Opcode list.
[2] Instruction also IFF_1 ← IFF_2

Flag Notation: • = flag is not affected, 0 = flag is reset, 1 = flag is set, ↕ = flag is set according to the result of the operation.

4.3 Glossary

Applet - An internet application that runs inside an internet browser.

Bank - A Computer science term used to describe a specific chunk of Random Access Memory (See Random Access Memory).

BIOS - Basic Input and Output System. A set of programs, addresses or routines inside RAM (See Random Access Memory) that provide certiain functionality for the computer system.

Bit - The smallest value used to represent computer data or memory in a base 2 binary numbering system having a value of 1 or 0.

Buad Rate - A term used to describe the ability of two devices ports (See Port) to establish a communication between them at a certain speed of data transfer.

Buffer - A permanent or temporary area of storage used to hold data.

Byte - A standard unit of measurement for computer data or RAM (See Random Access Memory)

CPU - The Central Processing Unit.

DOS - Disk Operating System.

I/O - Input and Output.

Interrupt - A term used to describe the need for a device or software program that must send a message to the Processor (See CPU) in order to gain its attention for useage.

Java - A programming language with internet and platform independent capibility.

Memory - Term used to describe an area of storage in a computer system. (See Random Access Memory, Bank, Buffer)

Memory Mapped - A term used to describe how a computer system connects it I/O (See I/O) to RAM (See Random Access Memory).

OP Code - The basic unit of instruction in a computer system. This is what is executed when a computer program is running.

Operating system - The software program that manages low level hardwareand software management inside a computer system.

Parallel - Data transmission that occurs in a side by side manor using multiple data lines to transmit data across a specific line.

Port - A term used to describe the means for I/O (See I/O) internally and externally in a computer system.

RAM - See Random Access Memory.

Random Access Memory - The second fastest form of storage used inside a computer system commonly used for application execution and data storage.

Register - The fastest form a storage inside a computer system usually constrained to a finite size depending on a particular system.

ROM - Read Only Memory. Usually contains useful programs or data for Operating System (See Operating System), hardware and program useage.

Serial - A term used to describe inline communication or data that is sent one after another either interanally or externally.

Virtual Machine - A term used to describe a software or hardware program that emulates a given environment in which its executing applications are thought to be running.