

# **Educational Personal Computer**

---

# Contents

<b>1</b>	<b>Goals</b>	<b>1</b>
1.1	Philosophy . . . . .	1
1.2	Audience . . . . .	1
1.3	Why a Simulator? . . . . .	1
1.4	Why EPC? . . . . .	2
<b>2</b>	<b>The EPC Architecture</b>	<b>3</b>
2.1	Consistency . . . . .	3
2.2	Data Types . . . . .	3
2.3	Instruction Set . . . . .	3
2.3.1	Jump Instructions . . . . .	3
	j . . . . .	3
2.3.2	Move Instructions . . . . .	5
2.3.3	Comparison Instructions . . . . .	5
2.3.4	Bit/Logic Instructions . . . . .	5
2.3.5	Arithmetic Instructions . . . . .	5
2.3.6	Input/Output Instructions . . . . .	5
2.4	Addressing Modes . . . . .	5
2.5	Registers . . . . .	5
2.6	Busses . . . . .	5
2.7	Memory Map . . . . .	5
2.8	I/O Addresses . . . . .	5
2.9	Exceptions . . . . .	5
2.10	DMA . . . . .	5
2.11	Disk . . . . .	5
2.12	Display . . . . .	5
2.13	Cache . . . . .	5
2.14	Virtual Memory . . . . .	5
2.15	Device Plugins . . . . .	5

---

# List of Tables

2.1	Data Types . . . . .	3
-----	----------------------	---

# Chapter 1

## Goals

### 1.1 Philosophy

Based on many years of experience teaching computer architecture and assembly language, EPC is designed with a few primary goals in mind:

- Maximize learning opportunities for students
- Minimize needless difficulty
- Simulate the development process as well as the machine

Learning opportunities are maximized by providing a feature-rich simulator, providing extensive feedback on each program run, and leaving as much as possible to students to do for themselves. The EPC simulator includes only essential macros for getting started with assembly language programming, such as basic terminal input and output. All other high-level software functions are intentionally left for students to design themselves, so they can experience "bare-metal" programming to the fullest.

Difficulty was minimized by creating an assembly language that is easy to learn and use and a simple, orthogonal machine code format.

The development environment requires use of a real assembler and macro preprocessor, so students become familiar with the process of generating machine code in the real world.

More details about EPC features are discussed in the sections that follow. [?].

### 1.2 Audience

EPC is a training tool for computer programmers and engineers.

It is meant for use in classes teaching assembly language, computer architecture, compiler design, and similar topics.

### 1.3 Why a Simulator?

Simulators have major advantages over real hardware as educational tools.

- Simulators allow the user to easily monitor the inner workings of a computer in as much detail as desired.
  - Hardware costs money. It must be purchased, maintained, upgraded and replaced.
  - A simulator, like any properly-written software, can be made portable, so that it can be used on virtually any operating system and hardware.
-

## 1.4 Why EPC?

- Modern commercial CPUs are designed primarily to be fast and cheap. Some of the most important goals are performance/cost and performance/watt. To accomplish these goals, sacrifices are often made to ease of use, leading to confusing instruction sets and inconsistencies in the hardware design. Hence, these architectures are not at all programmer-friendly.

This is necessarily not a bad thing in today's computer industry, since the only people typically affected by it are the handful of geniuses who write compilers.

However, for teaching a course in assembly language and computer architecture, a far more programmer-friendly instruction set is highly desirable. This will allow students to learn a great deal more about computer architecture in general, since they won't lose time struggling with an overly difficult language.

The EPC architecture is much easier to learn and use than most modern commercial architectures. The instruction set and machine code format are simple and orthogonal, but complete.

Students will not be overwhelmed with complexity and therefore can be exposed to all major computer architecture concepts such as memory hierarchy, addressing modes, etc. within the confines of a typical college semester.

- Free and open source.
  - The EPC simulator is written in ISO C and the assembler is written in ISO C++. Both are designed to be portable to any POSIX-compatible platform. This includes any Unix-like system, including Mac OS X, and MS Windows running Cygwin or any other POSIX compatibility system.
  - EPC more accurately simulates the programming process of real hardware. Programs are written in any text editor, then assembled to produce a machine-code "binary", which is executed by the simulator.
  - The machine code binaries are in the form of a human-readable "list" file, showing the machine code and assembly code side-by-side.
  - EPC reports a great deal of information about programs, such as the number of instructions executed, the number of clock cycles used, unaligned memory accesses, memory used, etc. This allows programs to be easily rated for objective quality measures such as speed and resource use.
-

## Chapter 2

# The EPC Architecture

### 2.1 Consistency

EPC machine code is 100% consistent.

Unlike some simulators, the EPC architecture is little-endian regardless of the underlying architecture. This means a small performance penalty when running on a big-endian computer, but the difference is not noticeable.

### 2.2 Data Types

Type	Suffix used in instructions	Description
byte	b	8-bit integer or raw bit data
short	s	16-bit integer or raw bit data
long	l	32-bit integer or raw bit data
quad	q	64-bit integer or raw bit data
float	f	32-bit IEEE floating point
double	d	64-bit IEEE floating point
address	a	32?-bit memory address

Table 2.1: Data Types

### 2.3 Instruction Set

#### 2.3.1 Jump Instructions

j

j label

opcode	arguments
00	aa

Jump unconditionally to address 'label'.

j	label	00	aa	
jl	label	01	aa	
ret		0e		
jeq	label	02	aa	z = 1
jne	label	03	aa	z = 0
jlt	label	04	aa	n = 1
jle	label	05	aa	n = 1 or z = 1
jltu	label	--		c = 0 (jnc)
jleu	label	06	aa	c = 0 or z = 1
jgt	label	07	aa	n = 0 and z = 0
jge	label	08	aa	n = 0 or z = 1
jgtu	label	09	aa	c = 1 and z = 0
jgeu	label	--		c = 1 (joc)
joc	label	0a	aa	c = 1
jnc	label	0b	aa	c = 0
jov	label	0c	aa	v = 1
jnv	label	0d	aa	v = 0

**2.3.2 Move Instructions****2.3.3 Comparison Instructions****2.3.4 Bit/Logic Instructions****2.3.5 Arithmetic Instructions****2.3.6 Input/Output Instructions****2.4 Addressing Modes****2.5 Registers****2.6 Busses****2.7 Memory Map****2.8 I/O Addresses****2.9 Exceptions****2.10 DMA****2.11 Disk****2.12 Display****2.13 Cache****2.14 Virtual Memory****2.15 Device Plugins**