# [6.205] Block Diagram Report - Orca

Kosi Nwabueze
kosinw@mit.edu

and

Haoran Wen
hranwen@mit.edu

25 October 2023

## 1 Abstract

LISP machines were the go-to general-purpose computers during the height of the artificial intelligence revolution in the 1960s and 1970s at the MIT AI Lab. Unfortunately, due to commercial failure and poor management at Symbolics, they quickly fell out of popularity. In the year 2023, we propose a modern re-imagination of the LISP machine: Orca. Orca is a general-purpose microcomputer featuring a 16-bit, specialized reduced instruction set machine that can execute compiled LISP code. Our system also includes external peripherals: a PS/2 keyboard, a UART serial controller, and a text-mode video controller. We seek to demonstrate that machines specialized for executing high-level, garbage collected, programming languages are still effective despite previous commercial failure.

## 2 Memory

In total, the computer will support 64KB of program BROM, 128KB of program BRAM, 2KB of video BRAM, 4KB of font BRAM, and 1KB of miscellaneous BRAM for MMIO. As of now, our plan is to use the block RAM located on the AMD Urbana Board and not use any of the external DDR3 memory available.

## 3 Processor

Hao will implement the processor architecture. The processor's role is to execute instructions in sequence. So far, no implementation has been finished; however, the following sections describe work that will need to be completed for the finished design.

### 3.1 Instruction Set Architecture

Both Kosi and Hao will work on designing an instruction set architecture (no design has been started so far). Since RISC-V architecture is taught in many classes throughout MIT's curriculum (namely 6.0004, 6.004, 6.175, 6.039, etc.), we opt to design an instruction set inspired by it. Our custom ISA will be based on a RISC machine with a 16-bit wide address, control, and data busses. In addition, the instruction set will include specialized instructions and registers for dealing with closures, heap management, and garbage collection.

### 3.2 Single-cycle Design

Hao will initially construct a single-cycle design of the processor complete with an instruction fetch unit, decoding logic, a multiply/divide unit, an ALU, and a register file.

### 3.3 Pipelining

Hao will work on pipelining the processor, implementing the classical 5-stage RISC pipeline complete with data, structural, and control hazard management. At this stage, if time permits, Kosi will implement hardware acceleration for garbage collection.

## 4 Peripherals

Kosi and Hao will implement peripheral devices that will interface. So far, no implementation has been finished; however, the following sections describe work that will need to be completed for the finished design.

### 4.1 Video Controller

The video controller will implement an 80x25 text-mode video interface using a *video* BRAM which contains an array of glyphs to display and a *font* BROM which describes the dot matrix on how each will be printed. The video controller's BRAM can be accessed through memory-mapped I/O through software. This video controller will work similar to how VGA text mode worked on old IBM PCs running MS-DOS.

### 4.2 UART Serial Controller

We will implement a serial controller that allows characters to be sent over UART from a terminal emulator and will be stored in a buffer that can be accessed by software through memory-mapped I/O.

### 4.3 PS/2 Keyboard Controller

We will implement a controller to interface with a keyboard using the PS/2 protocol. The sent characters and control registers will be stored in a region accessible from software through memory-mapped I/O. We will not implement an interrupt-based system since it would add significant complexity to the microcomputer, instead, we opt for the simpler polling interface where the software is responsible for checking any new instructions.

### 4.4 ESP32 Network Card (Stretch)

As a stretch goal, we will implement computer networking by interfacing with a WiFi-enabled ESP32 over UART. The ESP32 will handle all the low-level networking protocol details such as WiFi, TCP, and IP. Software can interface with the control registers and hardware buffers through memory-mapped I/O. The UART serial controller we wrote in a previous module that communicated with an external laptop can be reused to communicate with the ESP32.

# 5 Software

Kosi will be implementing the compiler which will translate LISP code into our custom RISC machine code. Kosi will also be implementing a minimal runtime which will handle garbage collection and interfacing with external peripherals. So far, no implementation has been finished; however, the following sections describe work that will need to be completed for the finished design.

## 5.1 Compiler

Kosi will implement a simple compiler in Python which compiles programs written in LISP into our custom RISC instruction set. The compiler will feature some techniques uncommon to compilers for procedural languages such as lambda lifting, which is necessary, to compile languages that feature closures into machine code. In addition, the processor and compiler will support tagging and hardware-assisted garbage collection.

## 5.2 Programs

Kosi and Hao will each develop multiple LISP programs to assess the functionality of the processor. The dialect of LISP targeted will be a minimal version of Scheme, namely, a small subset of the standard outlined in R5RS. Furthermore, Kosi will construct small demo programs that highlight the capabilities of the computer such as a Mandelbrot simulation, Diffie-Hellman key exchange, Conway's Game of Life, and Tower of Hanoi. As a stretch goal, Hao will design a very simple operating system capable of supporting fundamental computer hardware functions, including memory access and program execution.

# 6 Deliverables

Below we have sketched out a rough timeline of the deliverables needed to finish the project. Since delays are inevitable, we schedule more stretch goals toward the end of the project.

| Week | Kosi | Hao |
|---|---|---|
| Oct 26 - Nov 2 | ISA Design + Compiler | ISA Design + Processor |
| Nov 2 - Nov 9 | Compiler | Processor |
| Nov 9 - Nov 16 | Compiler | Processor |
| Nov 16 - Nov 23 | Video Controller | UART Serial Controller |
| Nov 23 - Nov 30 | Video Controller | UART Serial Controller |
| Nov 30 - Dec 7 | PS/2 Keyboard | ESP32 Network Card |
| Dec 7 - Dec 14 | Final Report | Final Report |

# 7 Conclusion

For Kosi's end of the project, the absolute minimum viable product will be a functional compiler that outputs executable machine code. For Hao's end of the project, the absolute minimum viable product will be a functional single-cycle processor. A reasonable objective for the whole project would be to get the video controller interface working as well. Stretch goals would include getting functioning networking working using an ESP32. Our microcomputer demonstrates the viability of

a processor architecture designed to execute high-level languages through processor implementation, hardware-assisted garbage collection, tagging, and peripherals.

# 8 Block Diagram

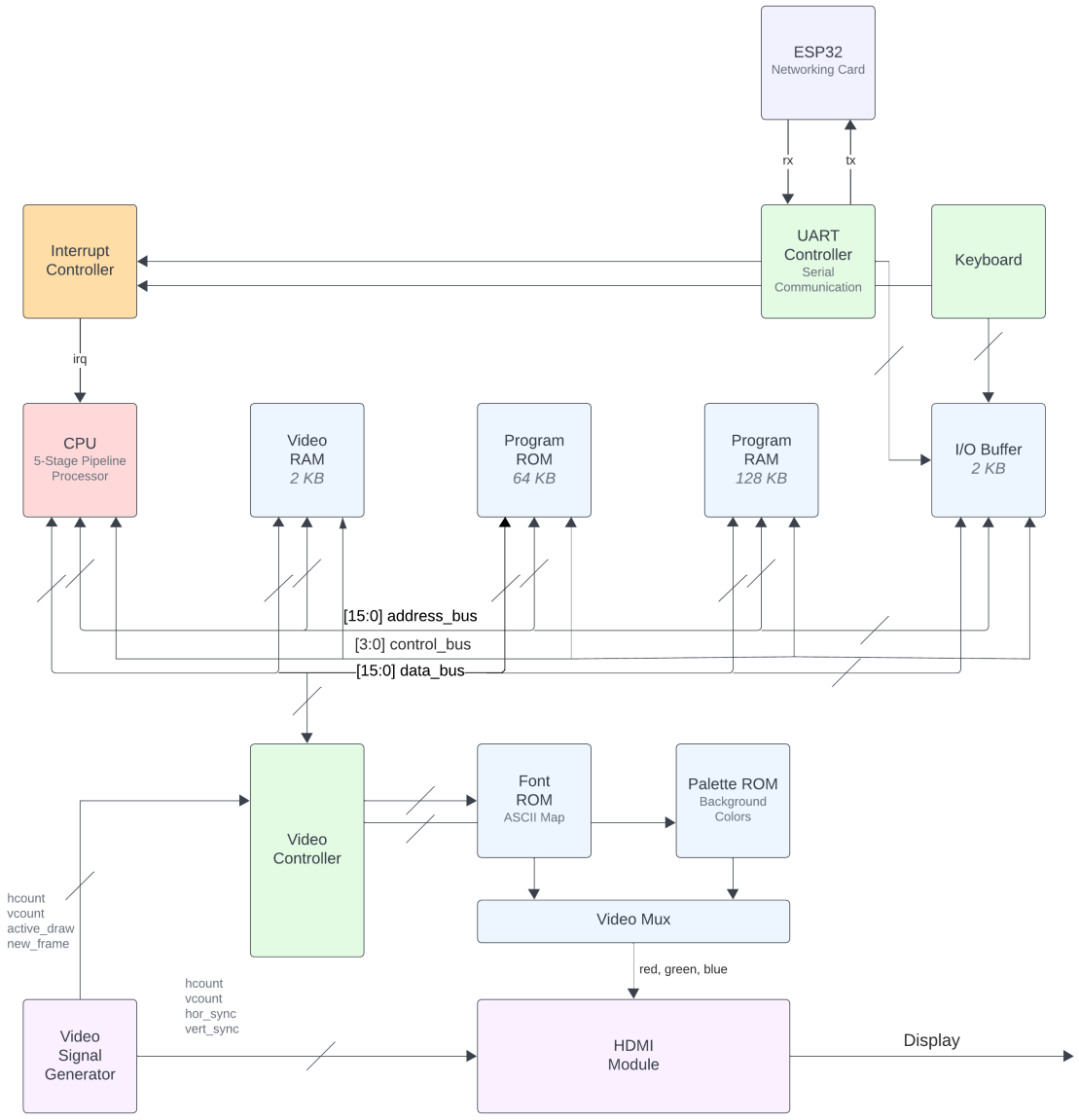


**Figure 1**: Microcomputer. The diagram shows RAM and ROM modules alongside a 16-bit processor along keyboard, video, and networking peripheral devices.