MIPS MIDI Player and Synthesizer
Devin Schwab, Diego Waxemberg, Dave Jannotta, Hirsch Singhal, Rob Meyer, Sam Castelaz
Final Report
May 2, 2014

Problem Statement

MIDI is a well-known audio format used to digitally simulate musical instruments. The format is well documented and perfectly suitable for a MIPS implementation. The MIPS MIDI Player and Synthesizer (MMPS) group aimed to create a proof of concept audio player to showcase the MIDI capabilities of a MIPS CPU.

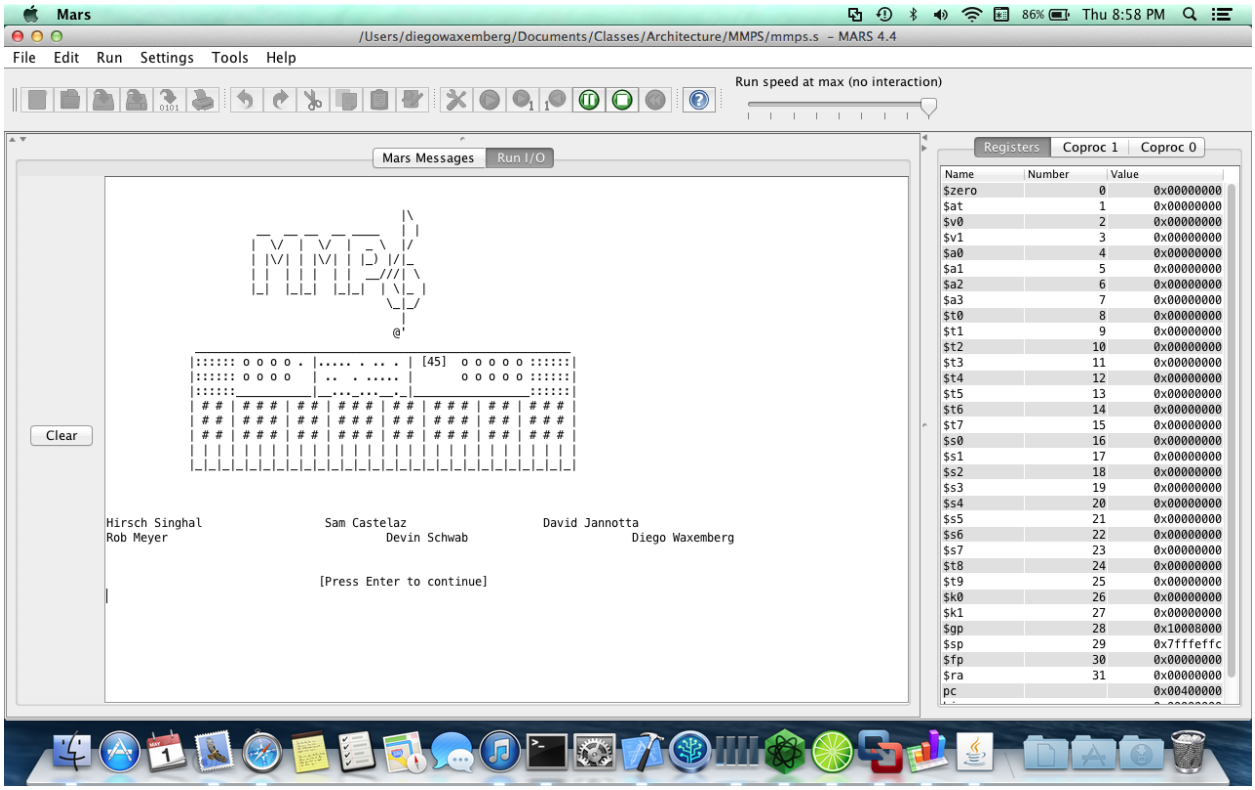
Major Challenges

There were several challenges that had to be overcome during the course of this project. One of the most important was playing the MIDI in MIPS. This challenge was overcome with relative ease however, due to a feature in the MARS simulator that allowed MIDI to be produced via a system call. This allowed us to request the simulator play the specified note and even had support for playing synchronously or asynchronously.

Another challenge was interpreting the user's input and storing that as MIDI events. This was solved by creating a custom interface built on top of the default MARS interface. By using custom defined keywords, the user's input could be parsed and interpreted.

Key Components

The MMPS can be broken down into 3 main components: the console, the storage, and playing.

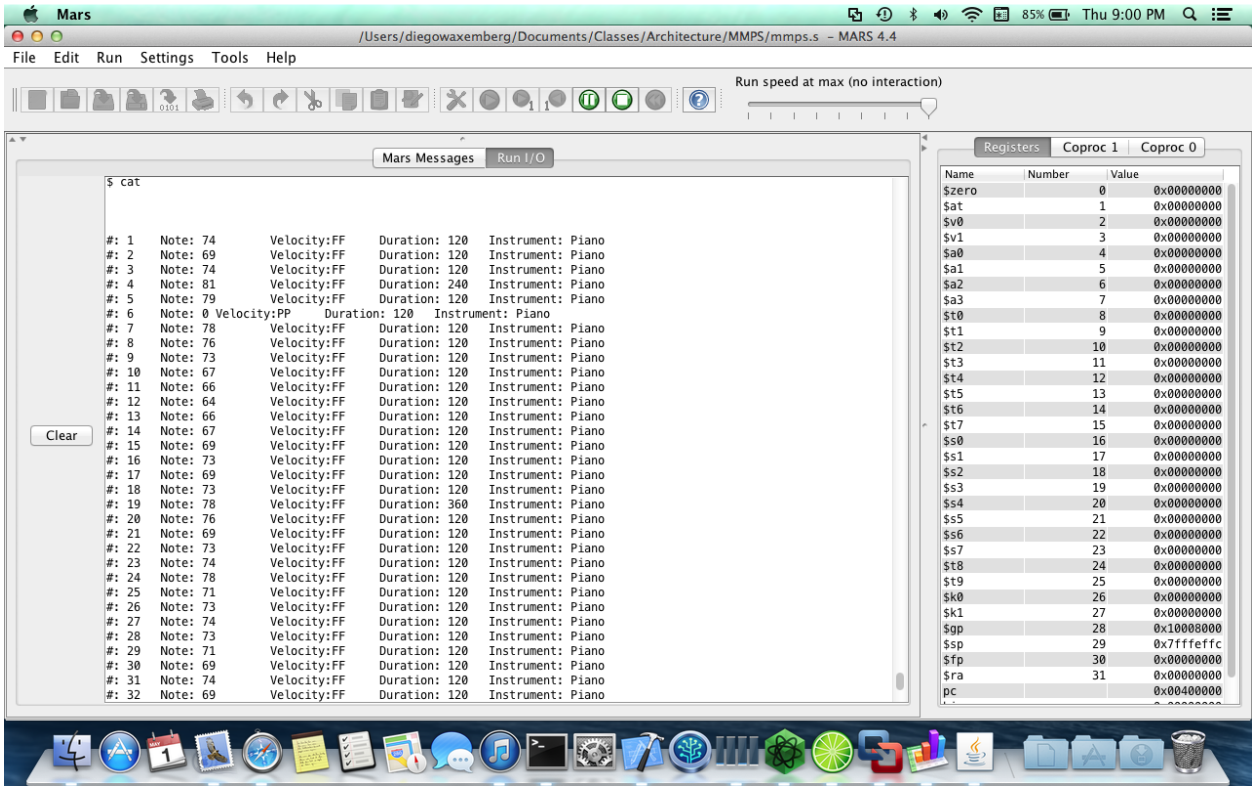


The program started in QtSpim which has a very basic input terminal. It did not support backspace when accepting input. To get around that the console sits in a loop. Each loop iteration it waits for a

character using the read char syscall. When the character is added then it it checked for being a backspace, enter or something else. If its backspace the last character in the input buffer is changed to null and the whole screen is redrawn. If it is enter the input buffer is parsed using a custom program DFA. If the command matches then the arguments are converted to the proper types and the command's function is called. If it is any other character then the entered character is appended the input buffer and the loop starts again. To redraw the screen blank lines are used to move the console down and then the saved strings are written using syscalls.

Storage was implemented using a dynamic memory allocator (sbrk) and insertion sort to place each MIDI message in the correct location. Each MIDI event was parsed to determine the absolute time that it would be executed and then using that time it was inserted into the array at the appropriate location. Since the array had to be increased by the size of 1 event (8 bytes) each time, the entire insertion process had O(N) run time as its worst-case. Events were shifted after each comparison with the new event and then the event was inserted in the proper location.

The dynamic allocator also accounted for allocating memory to store events read in from a file. The size of the file was passed to the allocator and the allocator would increase the array size by the difference between the currently allocated and the file size. The records were then stored in the array starting at the initial address, thus overwriting any previous values. This allowed for the least amount of memory to be allocated as possible.



Integration

The console and associated decision tree are used to tie the various components together. Each component is called by the console, runs, and then returns control to the console. In this fashion the only state saved between console calls is the size of the active record. MIDI on/off messages are generated using the parameters in the *add* command, in the following 8-byte format:

dd dd dd dd (time delta, in time ticks)

ci (command, instrument)
nn (note, 0-127)
vv vv (volume, 0-127)

The time delta and volume are encoded using 7-bit variable length quantities. The corresponding on and off messages are each stored in a pair of words for the memory allocator and record storage system to save. On and off messages are stored sequentially on the heap.

In order to play each MIDI message stored in the master record, the size of the dynamic memory is retrieved and used to iterate over each message. The record alternates on and off messages, where the first word of each message is the time delta between messages. The duration of the note is stored in the time delta of the off message, and the values for volume, note, and instrument are harvested from the messages. These parameters are provided to the MIDI syscall in MARS. Special note: the syscall in MARS expects a duration in milliseconds, but the duration is stored in time ticks, a MIDI construct. However, the tempo chosen commonly in MIPS and in our project is 480,000 microseconds/quarter note and 480 ticks/quarter note, yielding a conversion of 1 tick = 1 ms.

Saving and loading .mid files was performed using system calls and and a custom MIDI format for the project. The file consists of the length of the record in bytes followed by the entire master record. Upon loading, the length is read in and used to allocate the appropriate amount of memory. The rest of the file is then read in to this allocated space, reproducing the master record originally saved.

UI

The MMPS UI used the MARS console for user interaction. The user can select from a series of commands (see below) to manipulate the player. Devin Schwab built a second console on top of the MARS console to make it more user friendly. User-friendly additions included the ability to backspace, previous command log, error detection, and screen clearing to present a cleaner interface.

Commands:

- play (play the active record)
- play n v d i (play a single note)
- add n v d i (**n**ote, **v**olume, **d**uration in 16th notes, **i**nstrument)
- add d (**d**uration of a rest)
- cat (display a printout of each note in the active record)
- save f
- load f (f is the filename of the new MIDI file)

Specific Contributions

- Devin Schwab
  - Console design, logo, argument parsing, and demo tracks
- Diego Waxemberg
  - Dynamic memory allocator, record sorting, and record storage
- Dave Jannotta
  - Play functionality
- Hirsch Singhal
  - MIDI generation
- Rob Meyer
  - File save/load
- Sam Castelaz
  - Cat function