HW6 Design Document
Sarah Hall-Swan, Taylor Ampatiellos
Date: April 5th, 2016

OVERALL DESIGN/ARCHITECTURE:

***Overview of Program/Interactions between Components*:**

Our program's main will first read in a single pathname for a file which contains the machine instructions for our emulator to execute. After confirming that the file exists and is valid, the main function will call execute_instructions() from UM_execution.h. Here is an outline of the UM program once this function is called:

- Initialize registers to zero using initialize_UM() from UM_execution.h.
- Read in data from the provided file one instruction at a time, storing the word in the 0 segment.
- At each time step, an instruction is retrieved from the word in the 0 segment whose address is the program counter.
- The program counter is advanced to the next word, if any, and the instruction is then executed by calling the corresponding UM_instructions.h function.
- The segmented memory is managed by UM_seg_abstraction.h.

COMPONENTS

***Overview of Components*:**

We have packaged the functions into three components (not including the provided bitpack.h and Hanson data types) . UM_instructions.h contains the functions called to complete each of the 14 UM instructions. UM_seg_abstraction.h contains functions for creating/releasing/accessing the segmented memory. UM_execution.h keeps track of the registers and loops through the inputted program and calls all of the instruction functions.

***Components, their Architecture, and Descriptions*:**

- UM_seg_abstraction.h
    - Hanson Data Types (to be declared privately in .c file):
        - Seq_T segment_ids: Contains the segment ids currently in use as 32-bit words (uint32_t)
            - Invariants:

- Each word in this sequence will always be a valid key in the table of segments.

- Array_T segment: Each mapped segment will be a Hanson array containing 32-bit words (uint32_t)
  - Invariants:
    - The size of the array will remain at the size given when the segment is mapped.
    - Each element in the array will be a 32-bit word or 0.
- Table_T seg_memory: A table holds all of the mapped segments. The keys in the table are the segment ids (uint32_t) and the values are the segments (Array_T).
  - Invariants:
    - The number of segments will always be equal to or less than the number of instructions read in.

- void map(int size, uint32_t id)
  - Creates a new Array_T of length “size”, initializes elements to 0, and stores in the table at key “id”.
  - Stores “id” in the sequence “segment_ids” to keep track of mapped segments.
- void unmap(uint32_t id)
  - Frees the Array_T at key “id” of the table, and removes id from the sequence “segment_ids”.
- uint32_t load_word(uint32_t id, int offset)
  - Returns the word at the segment “id” at offset “offset”, by accessing the table at key “id” and returning the value of the array at index “offset”.
- void store_word(uint32_t id, int offset, uint32_t word)
  - Puts the word in the segment by storing it in index “offset” of the array in the table at key “id”.
- uint32_t program_counter(int index)
  - Goes to index element of segment 0 and returns the word at that index.

- UM_instructions.h
  - uint32_t Conditional _move(uint32_t A, uint32_t B, uint32_t C)
    - Sets value in register storing A to equal B if C != 0
    - If C != 0, then return B
    - If C == 0, then return A
  - uint32_t segmented_load(uint32_t B, uint32_t C)
    - Returns the memory in segment B, offset C

        - Calls load_word(B, C)
    - void segmented_store(uint32_t A, uint32_t B, uint32_t C)
        - Stores C in segment A, offset B
        - Calls store_word(A, B, C)
    - uint32_t add(uint32_t B, uint32_t C)
        - Returns (B + C) % 2^32
    - uint32_t multiply(uint32_t B, uint32_t C)
        - Returns (B x C) % 2^32
    - uint32_t divide(uint32_t B, uint32_t C)
        - Returns (B / C)
    - uint32_t nand(uint32_t B, uint32_t C)
        - return ~(B & C) [bitwise NAND]
    - void halt()
        - Stops computation
        - Frees all memory
    - void map_segment(uint32_t B, uint32_t C)
        - Calls map function from UM_seg_abstraction to create a new segment with space for C words, using the segment ID B
    - void unmap_segment(uint32_t C)
        - Calls unmap function from UM_seg_abstraction to release the segment with the ID C
        - Recycles the identifier C for later use
    - void output(uint32_t C)
        - Display the value in register c on the I/O device (must be between 0 and 255)
    - uint32_t input()
        - Returns the inputted value (must be between 0 and 255)
        - If EOF, returns a 32-bit word where every bit is 1.
    - void load_program(uint32_t B, uint32_t C)
        - Duplicates segment B and moves duplicate to segment 0
            - Unmaps segment 0 with unmap_segment(0)
            - Maps a new segment 0 the length of the segment B
            - Loops through segment B, loading the word from B and storing into segment 0
    - uint32_t load_value(uint32_t word)
        - Returns the value of the 25 least-significant bits in word
- UM_execution.h
    - Data Types:
        - 8-element local array for the registers
    - void execute_instructions(FILE *input)
        - Calls initialize_UM() before executing any instructions
        - Runs a loop through the program in segment 0 (for loop running until int i >= length of segment 0)

- Calls program_counter(i) during each iteration of the loop and is given the instruction at that line of the program, sets A, B, and C to appropriate registers
- With each instruction, calls the appropriate function in UM_instructions, passing the appropriate values in the registers defined in the 32-bit word
- Special instructions:
  - Load program (code 12) first sets the program counter (int i of the loop) to the value in register C, then calls load_program.
  - Load value (code 13) resets A to be the value in the register specified by the 3 bits after the opcode, then calls load_value.

- void initialize_UM()
  - Initializes the 8-element array to 0
  - Reads in the program and stores the instructions in segment 0
    - Loop: store_word(0, i, input_word) until end of input

TESTING

***Overview*:**

Along with this design doc submission you will find a .c and .h file which contains our unit tests for the segmented memory module. Below, you will find test explanations for each of the possible UM instructions -- our final submission will include coded unit tests for these as well.

***Testing of Instructions*:**

- uint32_t conditional _move(uint32_t A, uint32_t B, uint32_t C)
  - We will pass this function value combinations consisting of zero and nonzero values of C, then analyze the resulting return value. When C is nonzero, the return value should be B, and when C is zero, the return value should be A.
- uint32_t segmented_load(uint32_t B, uint32_t C)
  - If the segmented load refers to an unmapped segment or a location outside the bounds of a mapped segment, the machine fails (failure may be treated as an unchecked run-time error). We will test this function with varying inputs for C to ensure that the load runs for all locations within a mapped segment, and with unmapped input for B to make sure the function fails correctly.
- uint32_t add(uint32_t B, uint32_t C)
  - Run the program with various inputs and analyze the resulting return to ensure the program is calculating correctly.
- uint32_t multiply(uint32_t B, uint32_t C)
  - Run the program with various inputs and analyze the resulting return values to ensure the program is calculating correctly.
- uint32_t divide(uint32_t B, uint32_t C)

    - Run the program with various inputs and analyze the resulting return values to ensure the program is calculating correctly. If this function tries to divide by zero, the program fails.
- uint32_t nand(uint32_t B, uint32_t C)
    - Run the program with various inputs and analyze the resulting return values to ensure the program is calculating correctly.
- halt()
    - Run valgrind to ensure that all data is freed when the program is halted.
- map_segment(uint32_t B, uint32_t C) and unmap_segment(uint32_t C)
    - These functions directly call functions from UM_seg_abstraction, so they are tested in the unit tests for the segment abstraction.
- output(uint32_t C)
    - Compare this function's output to stdout to the C value it is given (should be the same).
- input()
    - Compare the resulting C return value to the input given to the function through stdin.
- load_program(uint32_t B, uint32_t C)
    - Check that the contents of B are being duplicated exactly and stored in segment 0.
    - Ensure that the program counter is set to segment 0 at offset C.
- load_value(uint32_t word, uint32_t A)
    - Ensure that the value being stored in register A matches the value in the 25 least-significant bits of word.

***Other Tests*:**

- To ensure that the program fails correctly:
    - Supply the UM main() with an incorrect/nonexistent filename.
    - Have the program pointer at the beginning of a machine cycle point outside the bounds of segment 0 .
    - Provide an incorrect/nonexistent instruction.
    - Have an instruction return a value larger than 255.
    - Have an instruction unmap segment 0 or a segment that is not mapped.
    - Have an instruction load a program from a segment that is not mapped.
- void initialize_UM():
    - Ensure that all registers are initialized to zero.