Aarti Paan, Dhruti Pandya, Matthew Posa, and Chuhui Qiu

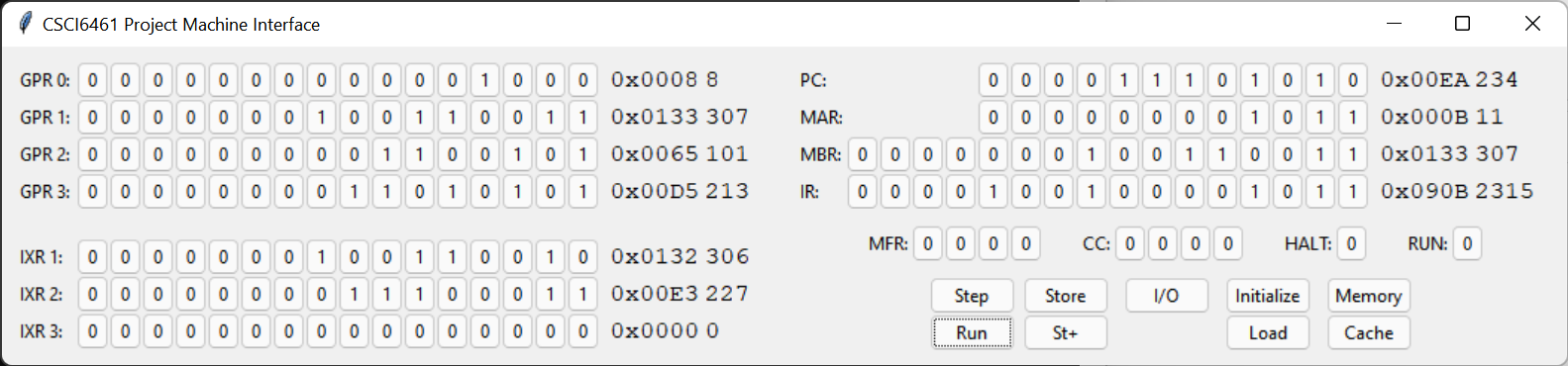
Professor Morris Lancaster

CS6461 Section 10 Spring 2022

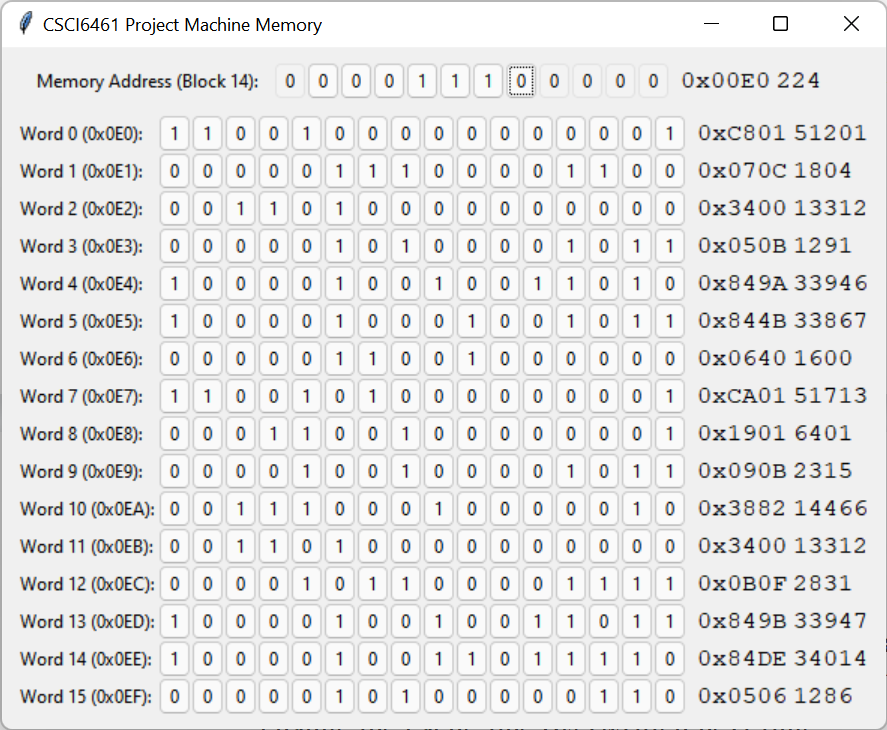
13 March 2022

**Group Project Phase 2 User Guide**

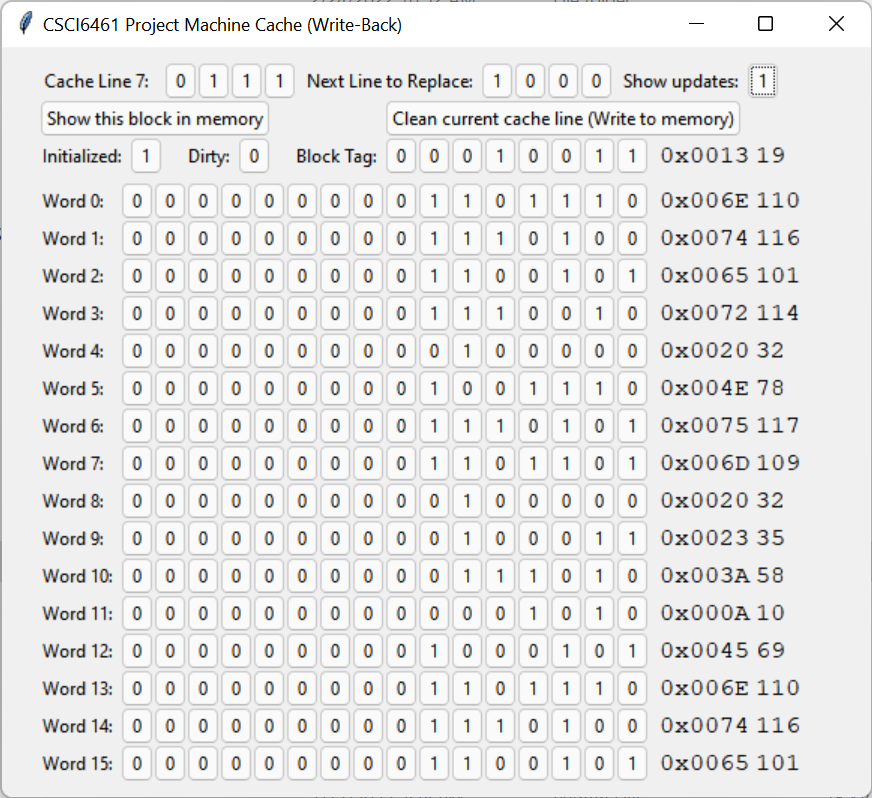
1. To start the simulator:
   1. If you are on Windows, double click “CSCI6461\_Project\_main.exe” to start the compiled executable.
   2. If you are not on Windows but have Python version 3.7 or higher installed, run “CSCI6461\_Project\_main.py” from Python to start the program.
2. Main Interface:



1. You should see an interface similar to the picture above.
2. All registers are big-Endian, meaning that bits on the left are the most significant bits.
3. Unlike other simulator implementations you may have seen, there are no dedicated “Switches” or “LD” buttons, instead, to change the value of a register, you simply click on the displayed “0” or “1” buttons to toggle them. The numbers on the right of the buttons are hexadecimal and decimal displays to help you read the current value of the register.
4. Not all registers are editable, some buttons are grayed-out, meaning they are read-only.
5. The GPR 0-3 are the “General Purpose Registers” numbering 0-3 to hold general data. The IXR 1-3 are the “Index Registers” numbering 1-3 to hold memory address index data.
6. The PC is the “Program Counter” which holds the address of the instruction to be executed.
7. The MAR is the “Memory Address Register” which holds an address to look up from the program’s memory.
8. The MBR is the “Memory Buffer Register” which holds data read from or to be written to the memory.
9. The IR is the “Instruction Register” which holds the instruction that is currently being executed or has been executed.
10. The MFR is the “Machine Fault Register” to warn user of faults:
    1. 0001 is “Illegal Memory Address: Reserved Location”, which occurs when a user tries to write to a reserved location.
    2. 0010 is “Illegal Trap code”; it is currently not implemented.
    3. 0100 is “Illegal Operation Code”, which occurs when an unknown operation is executed.
    4. 1000 is “Illegal Memory Address: memory installed”, which occurs when the memory address is beyond the machine’s memory range (0 to 2047).
11. The CC is the “Condition Code” for arithmetic and logical operations, it is set and used by different arithmetic, logical, and jump instructions.
12. The HALT is a “Machine Status Indicator”, it is 1 if the machine is halted (at which point no instruction can be executed). It can be manually toggled back to 0 to resume the machine.
13. The RUN is also a “Machine Status Indicator”, it is 1 if the machine is running. It can be manually toggled to continuously execute instructions, or to stop the ongoing execution.
14. The Step button is used to step the machine a single time, it would load the instruction pointed to by the PC to execute, then increment PC by 1 or transfer as defined by a transfer instruction.
15. The Run button is used to continuously execute instructions, this repeats until the machine halts. It can also be used to stop the ongoing execution.
16. The I/O button opens a window that allows the user to read program outputs or to input characters.
17. The Store/St+/Load/Initialize/Memory/Cache buttons are explained below.
18. How to Load/Store information from/into the memory: 3 different ways
    1. You may use any of the three methods listed below:
       1. The slow, “normal” way with MAR and MBR: First click on the buttons of MAR registers to set the memory word address you want to load/store from/to, then similarly set MBR if you want to store the data. Then you can click the “Load” button to load the data from memory[MAR] into MBR, or you can click “Store” to store data from MBR into memory[MAR]. “St+” is similar to Store, but it also increments MAR to allow repeated storing. This will first try to write into the machine's cache first, see section “Cache Interface” on how to make the cache write-back its content to the actual simulated memory.
       2. The faster way–“Initialize” by file: First click on the “Initialize” button, it will open a file prompt for you to choose a file. The file should be a plain text file (not rich text, json, xml, or word document), each line of the file should contain two hexadecimal numbers separated by a white-space character, the first number would represent a memory address while the second would contain the value to be stored. Once the file is chosen, the program will automatically read its content and write the memory as specified by the file. It will also reset all interface registers and the cache to 0. This can only store value into the memory, not load from them.
       3. The customizable, visualizable way with “Memory”: click on “Memory” Button, you would see a separate window open. See below.
19. Memory Interface
    1. The top of the window is a page navigation register in which you can choose the memory address you want to access. Then the 16 registers below will update to show the content of the next 16 words starting from the chosen memory location. You can click on any of the 16 lower registers to edit the words, and the changes will be immediately applied to the memory. (But not to the cache if a cache line of this block is already loaded, see section “Cache Interface”)
    2. The “Memory” Button on the main interface will be disabled if a memory interface window is already open, only a single memory interface window can be open at a time.



1. Cache Interface
   1. The simulated cache is of the write-back type. Write-through is not implemented.
   2. Cache Line x: Displays one of the 16 cache line numbers, if this is changed it will navigate to the new cache line.
   3. Next line to Replace: If the cache needs to write a new cache line (because of a cache miss), it will overwrite this cache line number, if this is changed it will change the cache line overwritten next time.
   4. Show Updates: if it is 1, every time a new cache line is loaded, it will automatically navigate the cache display window to that new line.
   5. Show this block in memory: When clicked, it will automatically open the memory interface and navigate to the memory address you are looking at in the cache. See section “Memory Interface” on how to read that.
   6. Clean current cache line (Write to memory): When clicked it will write everything in the current cache line to memory, then reset the "Dirty" bit to 0. Use this if you are loading memory with Store/St+ buttons on the main interface and want to write data to memory.
   7. Initialized: 0 if it contains invalid data (likely because has never been loaded with data from memory), 1 otherwise.
   8. Dirty: 1 if it contains data that has been modified but has not yet been written back to the memory, 0 otherwise.
   9. Block tag: the block number/tag number of this cache line, it is also identical to the first 12 bits of the address of the 16 memory entries currently on display.
   10. Word 0-15: the 16 entries of the block, they cache the values of memory locations from 0b[Block Tag]0000 to 0b[Block Tag]1111.



1. How to execute a program:
   1. Currently, 29 instructions are recognized (decimal opcodes are in parentheses): HLT(0)/LDR(1)/LDA(2)/LDX(3)/AMR(4)/SMR(5)/AIR(6)/SIR(7)/JZ(8)/JNE(9)/JCC(10)/JMA(11)/JSR(12)/RFS(13)/SOB(14)/JGE(15)/MLT(16)/DVD(17)/TRR(18)/AND(19)/ORR(20)/NOT(21)/SRC(25)/RRC(26)/STR(33)/STX(34)/IN(49)/OUT(50)/CHK(51)
   2. First, load the memory as instructed above.
   3. Then, make sure the HLT register on the interface is 0, toggle it if necessary.
   4. Next, manually click on the buttons of the PC register, so that it points to the beginning of your program in memory.
      1. In the case of the provided program 1, either 0x00, 0x0A, or 0x80 will run correctly.
   5. Then you may do either of the following:
      1. Clicking “Step” will step the simulated program exactly once, so one instruction will be executed, and PC will automatically increment by 1 to the next instruction (unless the previous instruction is a HLT)
      2. Clicking “Run” will repeatedly step the program, until a HLT instruction is executed.
   6. When a HLT instruction is executed, the HLT register on the interface will become 1 (“stopped”) and no more instructions will be executed.
   7. You may then inspect the registers directly, as well as the memory (with either Load button or the Memory Interface) to view the result of the execution.
2. What if the user made a mistake when operating the simulator?
   1. A “revert” functionality is planned but not implemented yet, the only existing way is to click “Initialize” to reset the memory and registers, to start again.
3. How to close the memory interface?
   1. Click the “x” on the top right corner of the memory interface window, all changes are saved in the memory.
   2. Closing the memory interface window will enable the “Memory” button on the main Interface window.
4. How to quit the simulator
   1. If any program is currently running, click on the “run” button, the “run” indicator, or the “halt” indicator to stop the program.
   2. Click the “x” on the top right corner of the main interface window; special handlers have already been set up to gracefully cleanup and quit the simulator.