**CPU Simulator (Phrase IV) Documentation**

**Design Notes**

This simulator program consists of twelve classes. These classes simulate different functions of the CSCI 6461 Computer.

1. **Control Panel**

This class is the main form of the CPU simulator project. It serves as the User Interface that simulates the console of the CS6461 Computer. The IPL button on the control panel runs the program. Switches are provided for user to input a specific program file. It shows basic registers and their current situations. I/O devices keyboard and printer are also embedded in the Control Panel. Field Engineer Console on the Panel provides useful information to examine the internal steps of the simulator while it is executing programs.

1. **Memory**

Memory has 8 banks in our simulator. Each bank is a 2-D array composed of 20-bit 256 words. The first index of the Memory array delegates the address value. The second index of the Memory array contains 20-bit content. The Memory Class has a sub-level cache that contains 16 blocks. When we want to read data from the memory or write data to the memory, the computer will check the cache memory first.

1. **Cache**

This class simulates Cache Design of the CSCI 6461 computer. Memory class instantiates a sub-level cache as L1. Cache size is determined by the memory as 16 blocks. Each block contains 4 words. DIRECT MAPPED placement/replacement, WRITE-BACK and WRITE-ALLOCATE on a write-miss are used in our design.

Below is a block structure:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2~4 | 5~6 | 7~19 | 20~39 | 40~59 | 60~79 | 80~99 |
| Valid | Dirty | Index | Offset | Tag | Word1 | Word2 | Word3 | Word4 |

|  |  |  |
| --- | --- | --- |
| Index | Name | Definition |
| 0 | Valid Bit | Check if the block updated |
| 1 | Dirty Bit | Indicate whether the data in a block is valid |
| 2~4 | Index Bits |  |
| 5~6 | Offset Bits | Select the word on the block |
| 7~19 | Tag | Address of the first word on the block |
| 20~39 | Word1 | Content of the first word |
| 40~59 | Word2 | Content of the second word |
| 60~79 | Word3 | Content of the third word |
| 80~99 | Word4 | Content of the four word |

The method Read () is designed to read data from Cache. If the address asked is on the Cache, the method returns the address and “Read Hit” will display on the console. If the address is not found in Cache (Read Miss), the method will bring the address from the Memory by using Direct Mapped (Fetch ()). In case that there is a UPDATE on the block to be swapped (check DIRTY-BIT), the method will save the changes on the memory (WRITE-BACK). The method Fetch () is designed to bring four words from the Memory to the Cache. The method Write () is designed to write data to Cache. If the address updated is on the cache, the method will find the cache, and update it (Write Hit).

1. **ALU**

This Class simulates the ALU component. It has five Register class instances: OP1, OP2, RES, RES1 and RESlong. OP1 and OP2 represent the operant registers inside the ALU. When these operant registers are set, an operator function is called. The result is saved in RES Register.

1. **Register**

This class simulates the Registers. All the registers such as General Purpose Registers, Index Registers are instantiated from this class. The bit length of register is required. The Boolean value determines whether the register will be showed on the GUI. The default is false and the name of register is non-visible.

1. **File**

This class simulates the Register File components. It has two methods. It returns the content of register file and loads the register file with the given value.

1. **RomLoade**r

This class simulates the ROM Loader. A boot loader is created to load instructions from the virtual card reader. It reads a boot program from the virtual card reader. After decoding the assembly instruction into the machine instructions, it stores the instruction in the memory.

1. **Instruction**

This class includes the Instructions Set Architecture of our simulator. All the instruction methods and some related methods are written in this class.

1. **Input**

This class simulates keyboard and let users provide input data. It supports both actual and virtual keyboard by using ASCII code for conversion. When users press a key on their actual keyboard or click a button on the virtual keyboard, the ASCII value of the key is loaded to an input buffer and status bit is set to data available. When the processor asks the input data, it returns the input buffer and set status to no data available.

1. **Output**

This class simulates the output (printer) of our simulator. It prints the data available on the screen.

1. **Decoder**

This class converts the assembly code into binary code for each instruction.

1. **Pipelined Instruction**

This class is designed to show pipeline implementation of our CSCI 6461 Computer. All instruction is run under the categories; instruction fetch, instruction decode and execute. By using timer and thread, it allows consecutive instructions to work in concurrently. It also provides functionality to see the internal step of each instruction. By setting the working process from the Control Panel class, internal process of the instructions can be seen step by step while program is working.