Computer Architecture

CPU simulator-Part I

1. **Function Description**

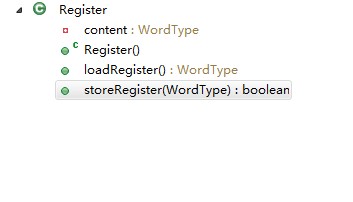
Generally, we design the CPU simulator using Java. We use classes to represent components of CPU and program each component’s function by class function. In Part I, we program a CPU simulator that can execute some instructions (STR, LDR). The simulator can execute these instructions one by one controlled by the buttons on the panel in UI. The basic principle of designing the CPU is to abstract each component of CPU into a Java class and divide the execution of instructions into separate steps so we can see change of memory and registers when a step is done.

1. **Basic Data Structure**

We use “int” to represent a 16-bits binary number. For instance we can represent 0000 0000 0001 0000(binary) using 16(decimal). Although, in Java a “int” number is stored as a 32-bits number, we add a restriction to ensure that there is no number larger than 1111 1111 1111 1111(binary) which is 65535(decimal). So we can use the bit operations in Java to operate these numbers.

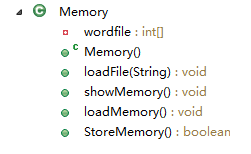
We use class WordType to represent 16-bits word. This class has class functions that can return different parts of a word. For instance, it can return the first six bits of a word which consist the opcode.

1. **Components** 
   1. **3.1 Register**



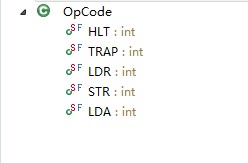
Content represents the word stored in the register. LoadRegister() loads the reigster and storeRegister stores a word into the register.

* 1. **3.2 Memory**



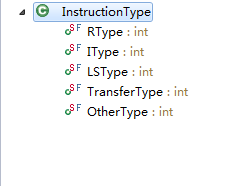
Wordfile stores all the word in a memory. LoadFile(string) loads a file of instructions into memory with the file name. ShowMemory()shows the content of memory. LoadMemory() loads a word from MDR into memory using address in MAR. StoreMemory() stores a word from MDR into memory using address in MAR.

**3.3 Opcode**



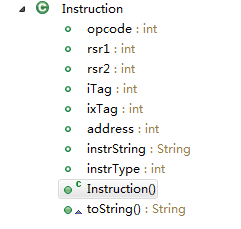
Some integers representing different opcodes.

* 1. **3.4 Instruction Type**



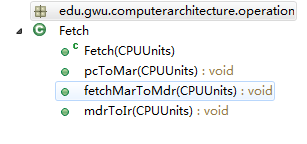
Some integers represent different instruction types.

* 1. **3.5 Instruction**



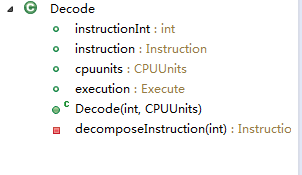
This is the class of instruction, defining several properties, such as opcode, iTag, ixTag.

**3.6 Fetch**



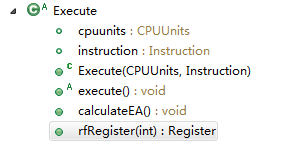
Fetch class has three methods: pcToMar which fetch the content of PC to MAR, fetchMarToMdr which fetch the content of MAR to MDR, and mdrToIr which fetch the content of MDR to IR.

**3.7 Decode**

****

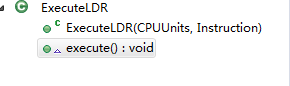
Decode class has member instructionInt which is the instruction in integer form, and instruction which is the refer to a instruction. Cpuunits which is a refer to the units of CPU, Execution which is a class executes do the detail execution of a instruction. DecomposeInstruction() is method that decompose a instructionInt to a instruction which means a integer into a instruction class. Each decode class do the decomposition for a instruction and build a execution class for the instruction.

**3.8 Execute**

****

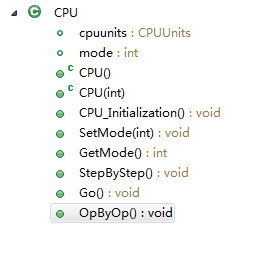
The execute class is an abstract class representing the basic execution that all the instructions will do including calculating the efficient address, choose the general registers. The abstract method execute which will be implement by the extended calls is the different parts of execution of instructions.

**3.9 ExecuteXXX**

****

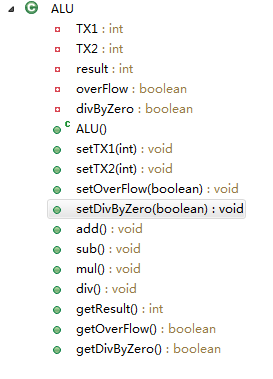
The ExecuteXXX classes represent all the different executions each instruction will do and it has the execute() method to do the execution. These classes are all extended from the Execute class.

**3.10 CPU**

****

Cpu class is the basic class representing the true CPU. It has two basic methods: OpByOp() which executes instructions instruction by instruction; Go() which executes instructions in one times.

**3.11 ALU**

****

The ALU class do the basic mathematical executions including add, sub, mul, div and we just use java operations to implement these operations.

1. **Designing**

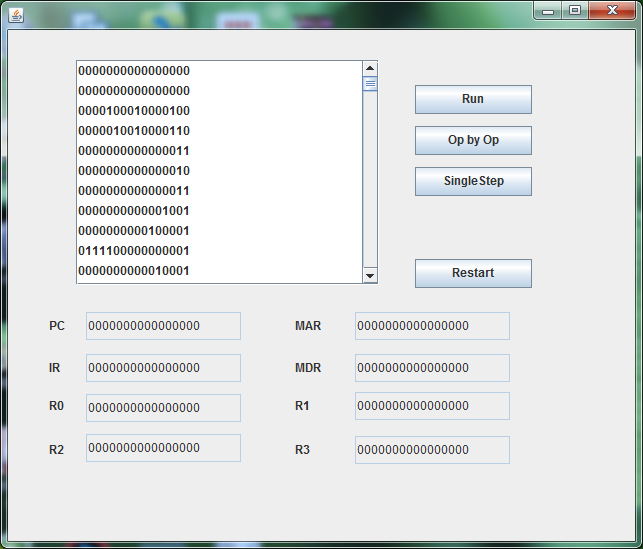
The following shows how we simulate a CPU

1. We build a CPU object. It has the CPUUnits class which includes all the registers and Memory.
2. We load a file of instructions into Memory
3. We put store the content of PC into mar
4. We build a Fetch object to fetch an instruction into MDR
5. We fetch the content of MDR into IR
6. We build a decode to decompose the content of IR into a instruction object
7. We use an execution object to execute the instruction object. The execution objects do different executions according to the instructions object.

Note: The fetch, decode execution objects all have a reference to CPUunits so that can change the content of CPU units.

The reason why we extract fetch decode and execution to be objects is that they are programmed by different members.

1. **UI**



The textfiled shows the content of each register. When the button “Run” is clicked the instructions will be executed until the simulator read a “HLT” instruction. When the button “Op by Op” is clicked the instructions will be executed instruction by instruction. And if we want to see every step in the execution of instructions, we can use the “SingleStep” button.

Last but not least, it’s the first time, our team members work together. We have a miscalculation of the due time. So this program lacks debugging that it has many bugs. We will correct the bugs in the future.