**Computer System Architecture Project**

**Implementation Features**

1. Microarchitecture: Harvard Architecture
2. Instruction Memory and Data Memory Size: 1024 x 32-bit
3. Cache: Direct Mapped Cache of Size 128
4. Total Number of Registers: 32 (32-bit) Registers

**Instruction Format**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| opcode  (5 bits) | rd  (5 bits) | r1  (5 bits) | r2  (5 bits) | immediate  (12 bits) |
| 31 27 | 26 22 | 21 17 | 16 12 | 11 0 |

**Instruction Set**

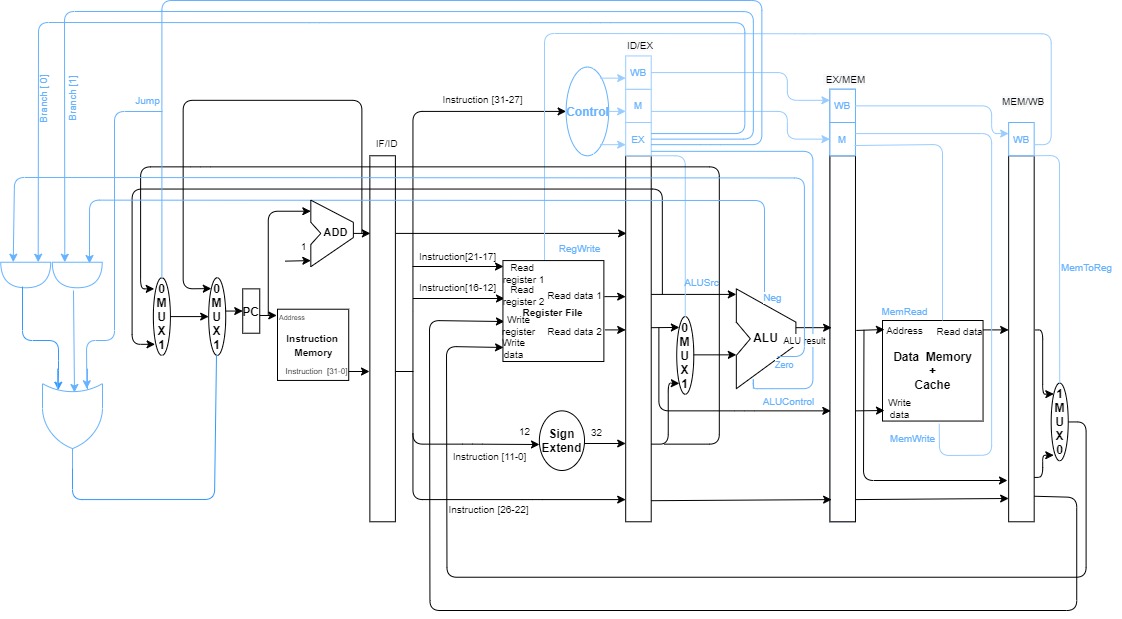
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NAME** | **MNEMONIC** | **OPCODE** | **EXAMPLE** | **OPERATON** |
| Subtract | sub | 00000 | sub rd r1 r2 | R[rd]=R[r1]-R[r2] |
| Add | add | 00001 | add rd r1 r2 | R[rd]=R[r1]+R[r2] |
| Add Immediate | addi | 00010 | addi rd r1 imm | R[rd]=R[r1]+SignExtImm |
| Multiply | mult | 00011 | mult rd r1 r2 | R[rd]=R[r1]\*R[r2] |
| Or | or | 00100 | or rd r1 r2 | R[rd]=R[r1]|R[r2] |
| And Immediate | andi | 00101 | andi rd r1 imm | R[rd]=R[r1]&SignExtImm |
| Shift Right Logical | srl | 00110 | srl rd r1 r2 | R[rd]=R[r1]>>R[r2] |
| Shift Left Logical | sll | 00111 | sll rd r1 r2 | R[rd]=R[r1]<<R[r2] |
| Load Word | lw | 01000 | lw rd r1 | R[rd]=M[R[r1]] |
| Store Word | sw | 01001 | sw r1 r2 | M[R[r1]]=R[r2] |
| Branch On Equal | beq | 01010 | beq r1 r2 imm | If(R[r1]==R[r2])  PC=SignExtImm |
| Branch On Less Than | blt | 01011 | blt r1 r2 imm | If(R[r1]<R[r2])  PC=SignExtImm |
| Set On Less Than Immediate | slti | 01100 | slti rd r1 imm | R[rd]=  (R[r1]< SignExtImm)? 1 : 0 |
| Jump Register | jr | 01101 | Jr r1 | PC=R[r1] |

**Registers**

There are 32 general purpose registers besides the pc and the pipeline registers. Their names and purposes are shown in the table below.

|  |  |  |
| --- | --- | --- |
| **NAME** | **NUMBER** | **USE** |
| $zero | 0 | The Constant Value 0 |
| $a0-$a5 | 1-6 | Arguments |
| $v0-$v1 | 7-8 | Values for Function Results and Expression Evaluation |
| $t0-$t9 | 9-18 | Temporaries |
| $s0-$s9 | 19-28 | Saved Temporaries |
| $sp | 29 | Stack Pointer |
| $fp | 30 | Frame Pointer |
| $ra | 31 | Return Address |

**Data Path**



**Code Flow**

Each module is represented in a separate class with its functionality: ProgramCounter, InstructionMemory, Control, RegisterFile, ALU, DataMemory (contains the cache as well). The class Processor contains 5 methods representing each stage: instructionFetch, instructionDecode, instrucionExecute, memoryAccess, writeBack. There are 4 hash tables that represent the pipeline registers: ifIdRegisters, idExRegisters, exMemRegisters, memWbRegisters. Each stage takes its needed input from those hash tables (except for the instruction fetch stage which takes its input directly from the ProgramCounter). Each stage sets its outputs as variables in class Processor which are then saved in the hash tables at the end of each clock cycle for the next stage to use them in the next clock cycle. The main method creates a new Processor instance, loads the program to the instruction memory, calls the 5 methods for each stage appropriately (for example in the first clock cycle it calls instructionFetch only, then instructionFetch and instructionDecode in the second clock cycle, etc.), handles shifting the contents of the hash tables at the end of each cock cycle, and prints the contents of the register file and data memory.

**Screenshots of the output**

The following are screenshots for the outputs for the following test program:

|  |  |  |
| --- | --- | --- |
| 00010000100000000000000000000101 | addi | r2 r0 5 |
| 01001000000010100111000000000000 | sw | r5 r7 |
| 00001000110010000110000000000000 | add | r3 r4 r6 |
| 00100010000100101010000000000000 | or | r8 r9 r10 |
| 01010000000000000010000000001010 | beq | r0 r2 10 |
| 00000010110110011111000000000000 | sub | r11 r12 r31 |
| 01000100010010100000000000000000 | lw | r17 r5 |

**Remark:**

To ease the testing process, each register in the register file is initialized to contain the 32-bit binary value equivalent to its number. For example, register 5 contains initially "00000000000000000000000000000101". The same is also the case in the data memory, each cell initially contains the 32-bit binary representation of its address. For example, cell 5 in the data memory contains initially "00000000000000000000000000000101". Their contents then change according to the program that will run afterwards.

