CSEn 601 Project Report

Instructor: Assoc. Prof. Hassan Soubra

Done by:

Team Molto (71)

**Contents**

1. Features…………………………………………………2
2. Registers………………………………………………...3
3. Cache……………………………………………………4
4. Instruction Format………………………………………5
5. Datapath…………………………………………….......6
6. Overview and Output………………………...…………7
7. Screenshots……………………………………………...8

**Features**

* Microarchitecture: Von Neumann architecture
* Instruction memory and data memory size: 1024x16 bit
* Total number of registers: 16 registers
* Instruction set: 2

**Registers**

1. General purpose

* $z: ($0), constant value 0
* $ja: ($1), return address
* $r0-$r9: ($2-$11), temporary data
* $m0-$m3: ($12-$15), for memory accessing

1. Specific purpose

* PC: program counter

**Cache**

1. Type: fully associative (directly mapped, each address maps to a single block and it can happen that two addresses map to the same block).
2. Replacement policy: in a way, we are using the last recently used method as if the block that matches the current index and tag is valid (meaning is already had data for another address), it is replaced with the new address.
3. Format: index[7-15] tag[0-6] valid[1 bit] data[mem(address)]
4. Size: 512 blocks

**Instruction Format**

1. R-Type: opcode[0-3] rs [4-7] rt [8-11] rd [12-15]

* add: [0000], rd = rs + rt
* sub: [0001], rd = rs - rt
* mul: [0010], rd = rs \* rt
* or: [0011], rd = rs || rt
* jr: [0100], PC = R[rs] where rt = [0000] & rd = [0000]

1. I-Type: opcode[0-3] rs [4-7] rt [8-11] immediate [12-15]

* andi: [0101], rt = rs && zero extend immediate
* addi: [0110], rt = rs + sign extend immediate
* slti: [0111], rt = rs < sign extend immediate? 1 : 0;
* lw: [0011], rt = M[rs + sign extend immediate]
* sw: [0100], M[rs + sign extend immediate] = rt
* beq: [1010], rs == rt? PC = PC + 2 + branch address
* blt: [1011], rs < rt? PC = PC + 2 + branch address

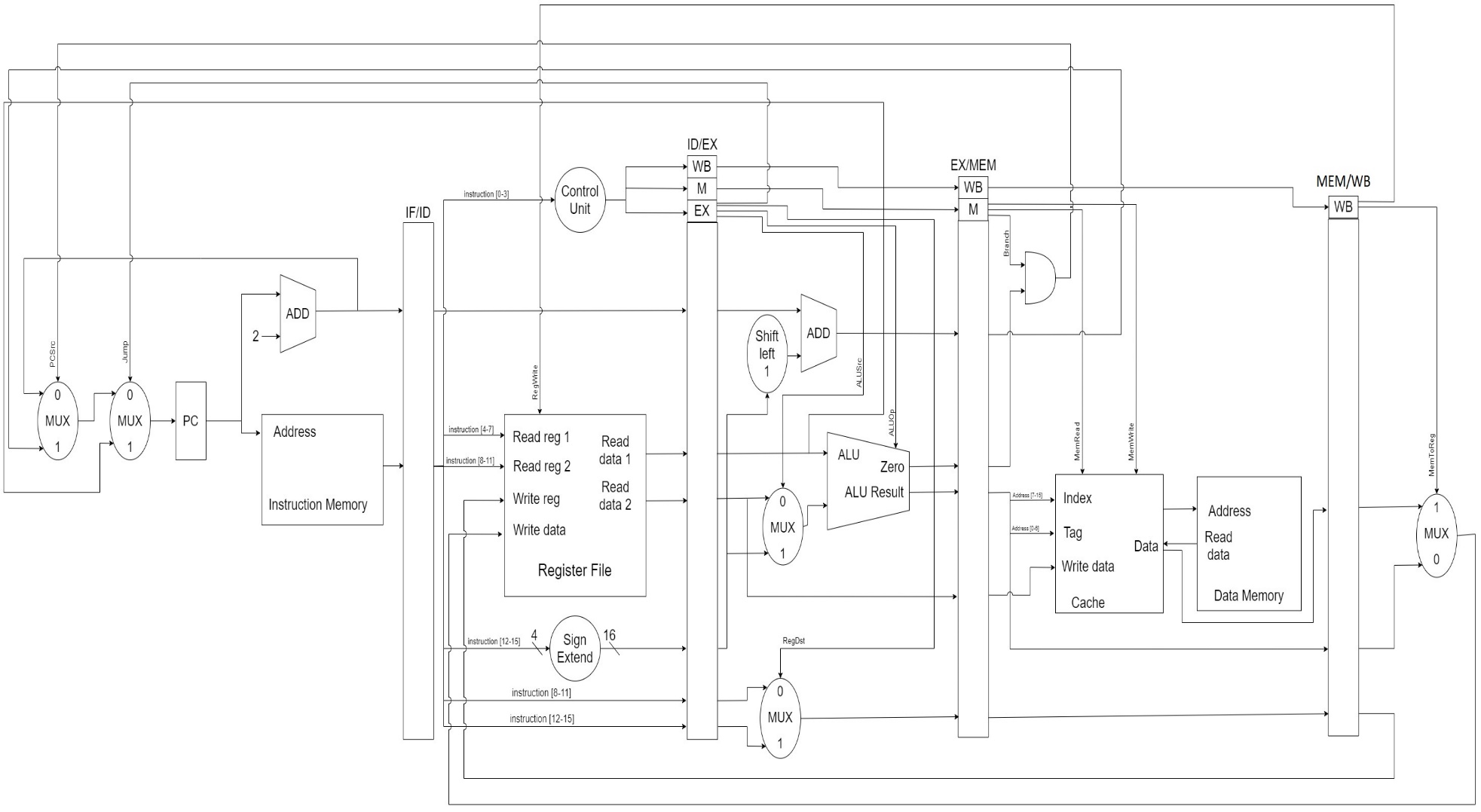
1. S-Type: opcode[0-3] rs [4-7] rt [8-11] shamt [12-15]

* srl: [1100], rt = rs << shamt
* sll: [1101], rt = rs >> shamt

Notes:

* zero extend: {12 (1b’0), immediate}
* sign extend: {12 (immediate 0), immediate}
* branch address: {11 (immediate 0), immediate, 1b’0}

**Datapath**



**Overview and Output**

Overview:

The process is broken down into five key stages; fetch, decode, execute, memory access, and writeback. Each of these stages is implemented in a separate class with all the variables being static. Each module within each stage is implemented in a separate class as well. The program is controlled using the CPU class which increments the PC according to the flags (PCSrc and Jump) and calls the main method of each of the five stages at each cycle. The memory, register file and cache are instantized in the CPU class. During each cycle, each stage will set the next stage’s instance variable statically so as to prepare them for the next cycle, consequently, we call the stages in reverse order each cycle so as to not overwrite the stages variables before it completes it’s function in the current stage.

Output:

For testing, I inputted all the 14 instructions in the instruction set as the program to make it easier and followed the exact format wanted. The screenshots are in the following page.

**Screenshots**

