Aidan Lambrecht

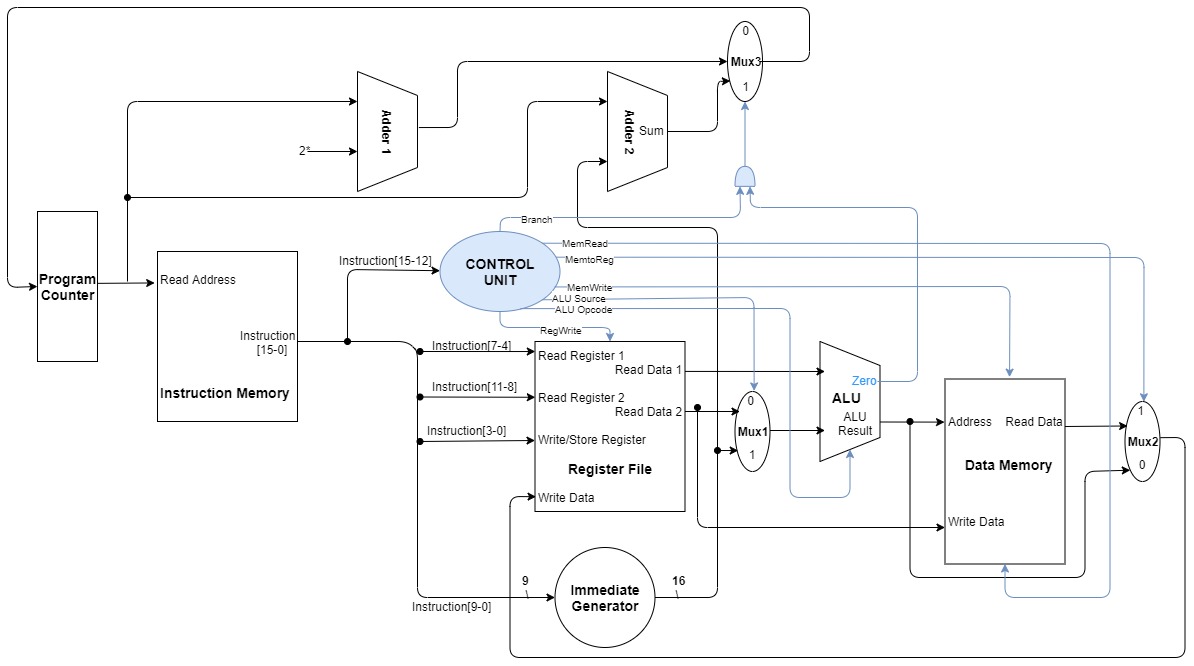
Dr. Harris

ELEC 5200

September 28, 2019

CPU Project Part Two: Datapath

**Datapath:**



**Components:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Inputs** | **Outputs** | **Function** |
| **ALU** | Read Register 1 (rs1), Mux1 | Zero flag bit, ALU Result | Takes in data from a register and either another register or the immediate generator to do the computations necessary for the CPU |
| **Register File** | Instruction Bits 11-0, Write Data from Data Memory | Data from two registers specified in the Instruction Bits | Holds all 16 registers available for use on the CPU and allows access to them for read/write purposes |
| **Program Counter** | Mux3 (an address for the next instruction) | Memory address of current instruction | Keeps track of the progress of the CPU through the given instructions saved in memory |
| **Instruction Memory** | Memory address of current instruction | Value of instruction at current instruction address | Allows access to instruction memory to provide the CPU with the necessary information to complete tasks |
| **Data Memory** | Address of memory to be accessed, data to be written to memory | Data read from memory at provided address | Allows access to data memory to store/load data |
| **Multiplexer 1** | Control signal, Read Register 2 data, Imm. Gen. data | Either the register data or the Imm. Gen. data | Provides ALU with the data necessary to current instruction execution, controlled by the Control Unit |
| **Multiplexer 2** | Control signal, Read Data from Data Memory, ALU result data | Either Data Memory data or ALU result data | Provides Register File with info necessary to write required data to a specified register, depending on the current instruction |
| **Multiplexer 3** | Control signal, incremented instruction address, branch/jump instruction address | Next instruction address | Determines the next address to send to the Program Counter, whether that be the next sequential address or another address specified by a jump or branch instruction |
| **Adder 1** | 2, current instruction address | Current instruction address + 2 | Prepares possible next instruction address for the Program Counter |
| **Adder 2** | Current address, immediate offset from Imm. Gen. | Current instruction address + immediate offset | Used during branch and jump commands to prepare the instruction address specified in the branch/jump |
| **Immediate Generator** | Current Instruction data | A 16-bit sign-extended immediate | Used in memory addressing and immediate operations |

**Fetching and Executing Instructions:**

Assume for all instructions (save HALT), register transfers begin with the PC sending the current instruction address to the Instruction Memory, followed by the Inst. Mem. sending pieces of the instruction to the Immediate Generator, Register File, and Control Unit. Or, in other words, Current Instruction (CI) = R[PC]

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | **Instruction** | **Register Transfers** | **Type** |
| 0000 | HALT | PC = PC | J |
| 0001 | LOAD | R[rd] = M[R[rd](9-0)+imm](15:0), PC = PC + 1 | L |
| 0010 | STOR | M[R[rd]+imm](15:0) = R[rd], PC = PC + 1 | L |
| 0011 | ADDR | R[rd] = R[rs1] + R[rs2], PC = PC + 1 | R |
| 0100 | ADDI | R[rd] = R[rd] + imm, PC = PC + 1 | I |
| 0101 | AND | R[rd] = R[rs1] & R[rs2], PC = PC + 1 | R |
| 0110 | OR | R[rd] = R[rs1] | R[rs2], PC = PC + 1 | R |
| 0111 | SHL | R[rd] = R[rd] << 0 | 1, PC = PC + 1 | I |
| 1000 | SHR | R[rd] = R[rd] >> 0 | 1, PC = PC + 1 | I |
| 1001 | JUMP | PC = PC + imm | J |
| 1010 | BRE | if(R[rs2]==R[rs1]) then PC = PC + imm, else PC = PC + 1 | B |
| 1011 | BRNE | if(R[rs2]!=R[rs1]) then PC = PC + imm, else PC = PC + 1 | B |
| 1100 | BRLT | if(R[rs2]<R[rs1]) then PC = PC + imm, else PC = PC + 1 | B |
| 1101 | BRGE | if(R[rs2]>=R[rs1]) then PC = PC + imm, else PC = PC + 1 | B |
| 1110 | SUBR | R[rd] = R[rs1] - R[rs2], PC = PC + 1 | R |
| 1111 | XOR | R[rd] = R[rs1] ^ R[rs2], PC = PC + 1 | R |

**Control Signals for Instructions:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ALU Operation** | **ALU Opcode 2** | **ALU Opcode 1** | **ALU Opcode 0** |
| No Operation | 0 | 0 | 0 |
| Add | 0 | 0 | 1 |
| Subtract | 0 | 1 | 0 |
| Shift Left | 0 | 1 | 1 |
| Shift Right | 1 | 0 | 0 |
| And | 1 | 0 | 1 |
| Or | 1 | 1 | 0 |
| Xor | 1 | 1 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Branch** | **MemRead** | **MemtoReg** | **MemWrite** | **ALU Source** | **ALU Opcode** | **RegWrite** |
| HALT | 0 | 0 | 0 | 0 | 0 | 000 | 0 |
| LOAD | 0 | 1 | 1 | 0 | 1 | 001 | 1 |
| STOR | 0 | 0 | 1 | 1 | 1 | 001 | 0 |
| ADDR | 0 | 0 | 0 | 0 | 0 | 001 | 1 |
| ADDI | 0 | 0 | 0 | 0 | 1 | 001 | 1 |
| AND | 0 | 0 | 0 | 0 | 0 | 101 | 1 |
| OR | 0 | 0 | 0 | 0 | 0 | 110 | 1 |
| SHL | 0 | 0 | 0 | 0 | 0 | 011 | 1 |
| SHR | 0 | 0 | 0 | 0 | 0 | 100 | 1 |
| JUMP | 1 | 0 | 1 | 0 | x | 000 | 0 |
| BRE | 1 | 0 | 1 | 0 | x | 000 | 0 |
| BRNE | 1 | 0 | 1 | 0 | x | 000 | 0 |
| BRLT | 1 | 0 | 1 | 0 | x | 000 | 0 |
| BRGE | 1 | 0 | 1 | 0 | x | 000 | 0 |
| SUBR | 0 | 0 | 0 | 0 | 0 | 010 | 1 |
| XOR | 0 | 0 | 0 | 0 | 0 | 111 | 1 |

**Discussion of Considerations:**

I chose a single-cycle datapath design because that was what we had learned in class at the time I began the project, because it was most easily understood from the book, and because upon talking to Dr. Harris, I learned that he would not dock us points for choosing a single-cycle design.

I have chosen edge-triggered registers due to the certainty it provides in knowing when operations will occur. Since there are no speed requirements for the CPU, I decided the time taken by edge-triggered registers was acceptable.

I chose to use two extra adders instead of routing everything through the ALU in order to speed up some of the adding operations dealing with address calculations. The rest of the miscellaneous components (the multiplexers) needed to be separate to prevent a mess of control signals.

Special consideration was given to the immediate generator and how it would decide to extend immediates correctly. It was decided that it would decide how to select bits based on opcode of the current instruction. Because of this, the generator was given all 32 bits of each instruction.