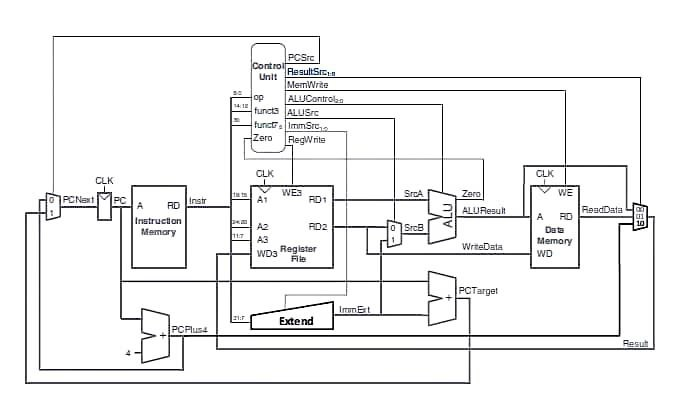
**RISC-V Single Cycle Processor**

The target of this project is the implantation of 32-bit RISC-V microarchitecture processor based on Harvard Architecture. The single-cycle microarchitecture executes an entire instruction in one cycle. However, the cycle time is limited by the slowest instruction.

In this architecture it provides the following set of RISC-V instructions:

* **R-Type:** add, sub, and, or
* **S-Type:** sw (store)
* **I-Type:** addi (add immediate), andi (and immediate), ori (or immediate), lw (load)
* **B-Type:** beq (branch if equal), bne (branch if not equal), blt (branch if less than)
* **J-Type:** jal (Jump and Link)



**Figure 1:** Single cycle processor RISC-V.

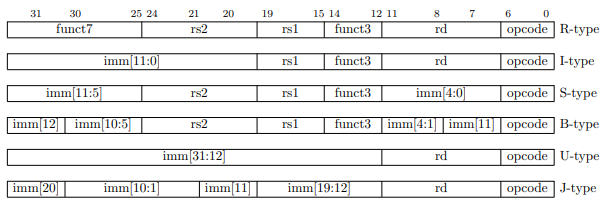
Let’s start to implement the design from the left, firstly the **program counter** is used to track the address of the next instruction to be executed. The multiplexer is used to select if the **PC + 4** is used as it is the next address as every cycle, the PC needs to be incremented by 4 (32 bits = 4 bytes) or the **PCTarget** is used, this will be used when jump, branch instructions are used.

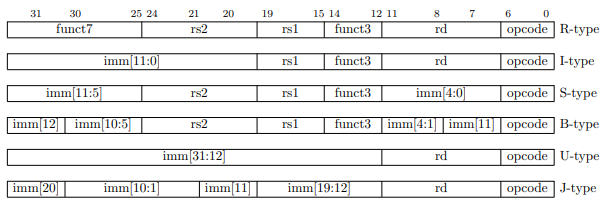
Simply the Multiplexer chooses between two options, the selection line is PCSrc.

|  |  |
| --- | --- |
| **PCSrc** | **PCNext** |
| 0 | PC + 4 |
| 1 | PCTarget 🡪 PC + ImmExt |

The PC is simply connected to the address input of the **instruction memory**. The instruction memory fetches the 32-bit instruction.

According to the type of the instruction it will be decoded in the **register file**:



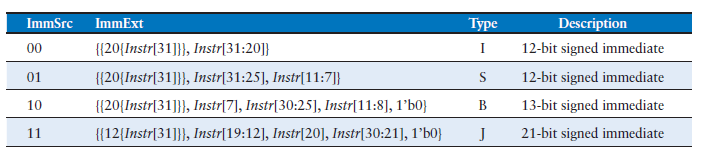


The **ALU** (Arithmetic Logic Unit) will apply the required operation according to instruction if required, this will be depended on **ALUControl**

|  |  |
| --- | --- |
| **ALUControl** | **Function** |
| 000 | SrcA + SrcB |
| 001 | SrcA - SrcB |
| 011 | SrcA & SrcB |
| 010 | SrcA | SrcB |
| 101 | SrcA < SrcB ? |

**The Extended Unit:**

some instructions also require an offset. The offset is stored in the immediate field of the instruction. It is a signed value, so it must be **sign-extended** to 32 bits. Sign extension simply means copying the sign bit into the most significant bits. By using ImmSrc, to select bits of immediate according to the type of instruction.



**Data Memory:**

* Has a single read/write port.
* If write is enabled, WE, is asserted, then it writes data WD into address A on the rising edge of the clock.
* RD is read not depend the clock edge.
* A is the memory address from which the data are read through the output port RD.

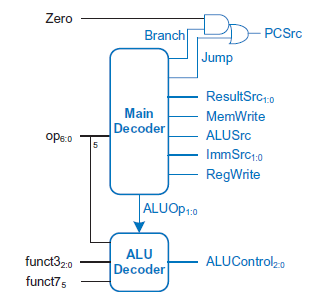
**Mux 3x1:**

To select the output is back to register file from memory or the result of alu operation.

Or if jump and link is used, you need to save the next instruction ( PC +4 ) to return to it again.

|  |  |
| --- | --- |
| **ResultSrc** | **Output** |
| 00 | ALUResult |
| 01 | ReadData |
| 10 | PC + 4 |

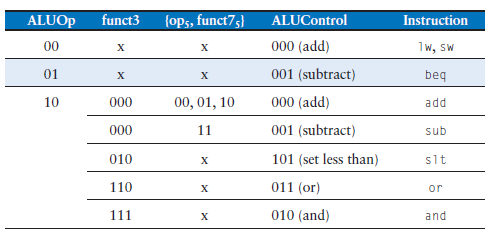
**Control Unit:**



**Truth Table of Main Decoder:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Opcode** | **Instruction** | **RegWrite** | **ImmSrc** | **ALUSrc** | **MemWrite** | **ResultSrc** | **Branch** | **ALUOP** | **Jump** |
| 3 | lw | 1 | 00 | 1 | 0 | 01 | 0 | 00 | 0 |
| 35 | sw | 0 | 01 | 1 | 1 | xx | 0 | 00 | 0 |
| 51 | R-type | 1 | xx | 0 | 0 | 00 | 0 | 10 | 0 |
| 99 | beq | 0 | 10 | 0 | 0 | xx | 1 | 01 | 0 |
| 19 | addi | 1 | 00 | 1 | 0 | 00 | 0 | 10 | 0 |
| 111 | jal | 1 | 11 | x | 0 | 10 | 0 | xx | 1 |

**Truth Table of ALU Decoder:**



**Assembly code for testing:**

main: addi x2, x0, 5 🡪 initialize x2 by 5

addi x3, x0, 12 🡪 initialize x3 by 12

addi x7, x3, -9 🡪 x7 = x3 + (-9) = 12 -9 =3

or x4, x7, x2 🡪 x4 = x7 | x2 = 3 or 5 = 7

and x5, x3, x4 🡪 x5 = x3 & x4 = 12 and 7 = 4

add x5, x5, x4 🡪 x5 = x5 + x4 = 4 + 7 = 11

beq x5, x7, end 🡪 x5 – x7 = 11 – 3 = 8 != 0, so it shouldn’t be taken

slt x4, x3, x4 🡪 x4 = x3 < x4 = 12 < 7 =0

beq x4, x0, around 🡪 x4 – x0 = 0 - 0 = 0, so it should be taken

addi x5, x0, 0 🡪 shouldn’t be done

around: slt x4, x7, x2 🡪 x4 = x7 < x2 = 3 < 4 = 1

add x7, x4, x5 🡪 x7 = x4 + x5 = 1 + 11 =12

sub x7, x7, x2 🡪 x7 = x7 – x2 = 12 – 5 =7

sw x7, 68(x3) 🡪 address 68 + 12 = 80 🡪 7

lw x2, 80(x0) 🡪 x2 = [80] = 7

jal x3, end 🡪 save the address of next instruction to return to it (x3 = 64)

addi x2, x0, 1 🡪 shouldn’t be done

end: add x2, x2, x3 🡪 x2 = x2 + x3 = 7 + 64 = 71

sw x2, 84(x0) 🡪 [84] = 71

**The generated hex file to be used in instruction memory**

00500113

00c00193

ff718393

0023e233

0041f2b3

004282b3

02728663

0041a233

00020463

00000293

0023a233

005203b3

402383b3

0471a223

05002103

008001ef

00100113

00310133

04202a23

**Simulation Result:**

