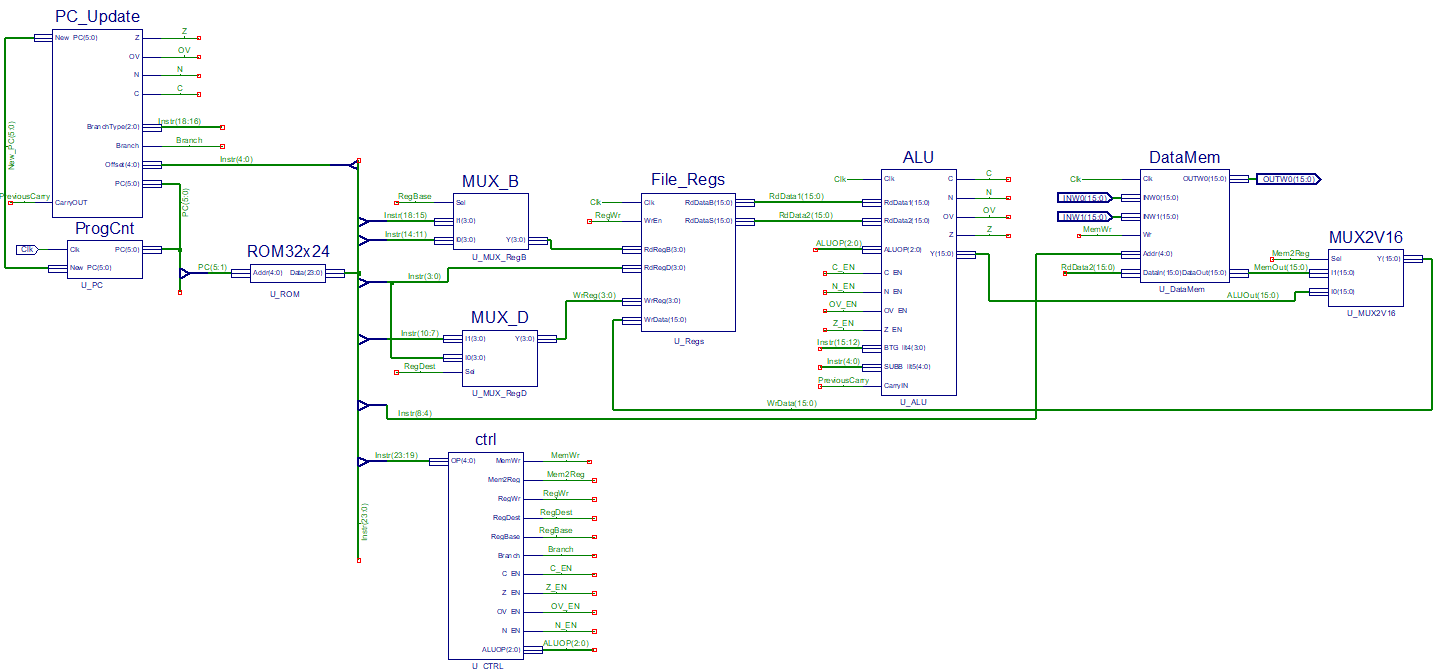
Pic24 Report

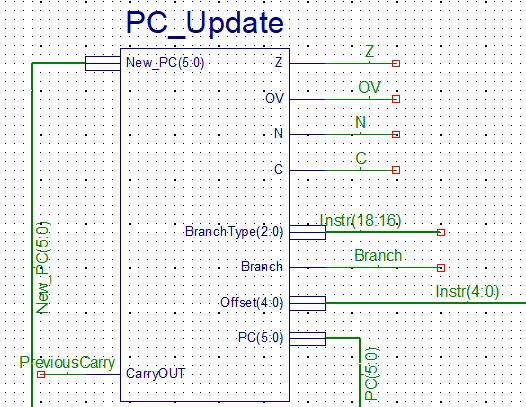
Name: Balan Adelin-Cristian

Project no: 5

The schematic of the microprocessor:

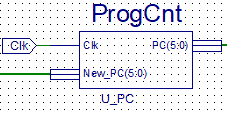
Components of the project:

**1. PC\_Update:**



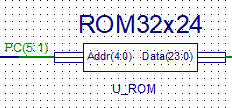
This block updates the program counter. If a branch instruction is involved, the increment of the program counter is done by adding the default step to the counter, 2, and the offset, which is shifted to the left, PC + 2 + offset \* 2. If no branch instruction is involved, the increment is done by adding the default step to the counter, PC + 2. The offset is parsed from the bits 15-0, but we can use 5 bits as well.

**2. Program Counter:**

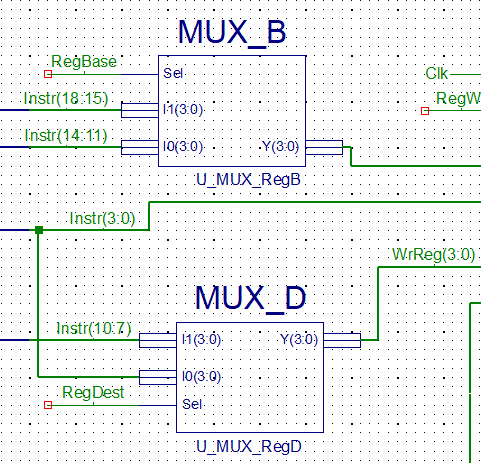


* This block is used to update the current program counter on a rising edge of the clock.

**3. Rom 32x24:**



* This block is the actual memory of the microprocessor. It contains a 32 words rom with 24 bits each.
* The data is sent through the Data pin using a selection based on the Addr’s pin value.

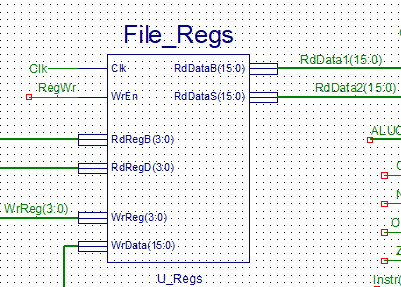
**4. MUX\_B and MUX\_D**

Blocks used to select the base and the destination registers.

The destination registers can be found between the bits 10-7 and 3-0.

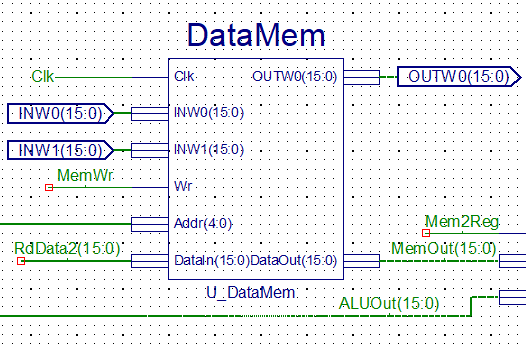
The base registers can be found between 18-15 and 14-11.

**5. File\_Regs:**

****

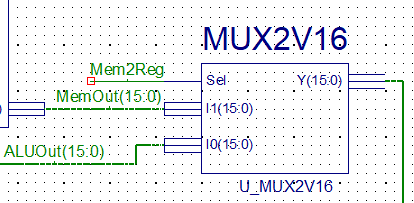
This block contains 16 registers(W0-W15), each with a length of 16 bits. Here we write the data from the operations which have a destination register and read the information for the ones with base or source registers.

**6. Data\_Mem**



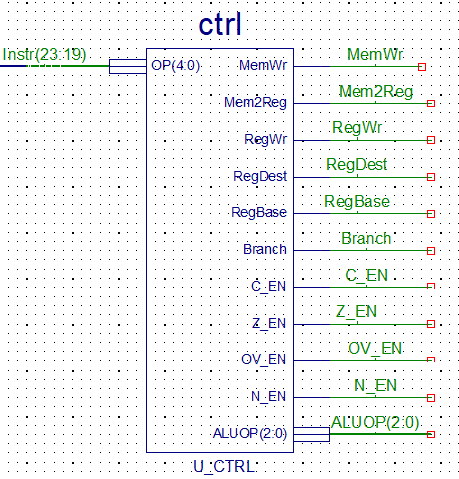
This block contains a 16x16 RAM module. This is the place where the MOV Wns, f instruction writes. The reading is done from the addresses 1020h(INW0) and 1022h(INW1) and the writing is done at the address 1024(OUTWO). The output DataOut will have the information from MemData, which is a value selected from the memory based on the Addr’s value, INW0 or INW1.

**7. MUX2V16**

****

This MUX decides, based on the Mem2Reg signal if the data is written in the register from the ALU bloc or from the memory (MOV f, Wnd).

**8. Ctrl**

****

Here are the signal used in all the other blocks.

For example, for the ADD Wb, Ws, Wd we have:

N\_EN = 1 (the negative flag is active)

OV\_EN = 1 (the overflow flag is active) Signals used in ALU for carry, zero status,

Z\_EN = 1 (the zero flag is active) overflow or negative status writing

C\_EN = 1 (the carry flag is active)

MemWr = 0 (No memory write)

Signal used in DataMem to enable the memory writing.

The only instruction which writes in the memory is MOV Wns, f.

Mem2Reg = 0 (No memory to registry write)

Signal used in MUX2V16 for the selection of the source of the data which is written in the register.

1 for ALU reading and 0 for Memory reading.

RegWr = 1 (Registry write is active)

Signal used in File\_Regs to enable the writing in the registers.

Branch = 0 (No branch enabled)

Signal used in PC\_Update to enable the branch jump.

RegBase = 1 (We have base register)

Signal used in MUX\_B for the selection of the base bits. The selection is done from the bits

18-15 of 14-11.

RegDest = 1 (We have destination register)

Signal used in MUX\_D for the selection of the destination bits. The selection is done from the bits 10-7 or 3-0.

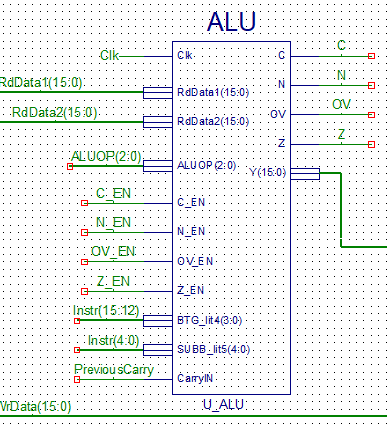
Other signals:

ALUOP – Selection id of the current instruction.

Signal used in ALU to select the current instruction.

OP – The opcode of the current instruction

**9. ALU:**

****

Here is done the actual processing of the arithmetical and logical instructions. The result is computed, based on the ALUOP from the CTRL block, using the RdData1 (base register) and RdData2 (source register).

BGT\_lit4 is the address of the bits involved in the #lit4 instruction.

BTG toggles the bit on the location #lit4.

My idea was to concatenate the bits from 15 to conv\_integer(BTG\_lit4) + 1 with the complemented bit and with the bits from conv\_integer(BTG\_lit4) – 1 to 0. For the cases when the value of the BTG\_lit4 is 0 or 15, the approach is similar, but with less concatenations.

SUBB\_lit5 is the address of the bits involved in #lit5.

SUBB involves the sum between the base register and the 2’s complement of the #lit5 and then the subtraction of the carry (the carry from the last operation). The carry is stored in PC\_Update from the previous instructions and transmitted to ALU.

LSR represents the shift to the right of the bits of the input (base register).

When the shift value is greater than 15, the result is automatically x”0000”.

If the shift value is less than 15, the result is computed by adding the bits between 15 and the conv\_integer(shift value) to x“0000”.

The NEG instruction computes the 2’s complement of the input.

This is done by adding the bit by bit complement of the input to x“0000” and then by adding 1 to the result.

The common instructions involve just simple operations which are done with the help of the +, -, AND and OR overloads in the compiler.

All the operations are computed on 17 bits for an easier carry computations.

The carry is the 17th bit of the result.

The zero status is just a comparation between the least significant 16 bits of the result and x”0000”.

When result(15:0) is x”0000” the zero flag is activated.

The overflow flag checks if the operations make sense. For example, if we want to compute the sum and the sign of Wb is 1 and the sign of Ws is 1 and the result is 0, the overflow flag is activated.

For the SUB and SUBB the exceptions are “” – “” = “”.

For NEG, the exception is “” = “”.

The negative status checks the 16th bit of the result.

**Instruction coding:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Coding | 2222 3210 | 1111 9876 | 1111 5432 | 1100 1098 | 0000 7654 | 0000 3210 | Flags |
| ADD Wb, Ws, Wd | 0100 | 0www | wBqq | qddd | dppp | ssss | N, OV, Z, C |
| SUB Wb, Ws, Wd | 0101 | 0www | wBqq | qddd | dppp | ssss | N, OV, Z, C |
| AND Wb, Ws, Wd | 0110 | 0www | wBqq | qddd | dppp | ssss | N, − , Z, − |
| IOR Wb, Ws, Wd | 0111 | 0www | wBqq | qddd | dppp | ssss | N, − , Z, − |
| MOV f, Wnd | 1000 | 0fff | ffff | ffff | ffff | dddd | none |
| MOV Wns, f | 1000 | 1fff | ffff | ffff | ffff | ssss | none |
| BRA expr | 0011 | 0111 | nnnn | nnnn | nnnn | nnnn | none |
| BRA C, expr | 0011 | 0001 | nnnn | nnnn | nnnn | nnnn | none |
| BRA OV, expr | 0011 | 0000 | nnnn | nnnn | nnnn | nnnn | none |
| BRA Z, expr | 0011 | 0010 | nnnn | nnnn | nnnn | nnnn | none |
| BRA N, expr | 0011 | 0011 | nnnn | nnnn | nnnn | nnnn | none |
| LSR Wb, Wns, Wnd | 1101 | 1110 | 0www | wddd | d000 | ssss | N, − , Z, − |
| NEG Ws, Wd | 1110 | 1010 | 0Bqq | qddd | dppp | ssss | N, OV, Z, C |
| BTG Ws, #bit4 | 1010 | 0010 | bbbb | 0B00 | 0ppp | ssss | none |
| SUBB Wb, #lit5, Wd | 0101 | 1www | wBqq | qddd | d11k | kkkk | N, OV, Z, C |

**CTRL signals:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Coding | OPCODE | N\_EN | OV\_EN | Z\_EN | C\_EN | ALUOP | MemWr | Mem2Reg | RegWr | Branch | RegDest | RegBase |
| ADD Wb, Ws, Wd | 01000 | 1 | 1 | 1 | 1 | 000 | 0 | 0 | 1 | 0 | 1 | 1 |
| SUB Wb, Ws, Wd | 01010 | 1 | 1 | 1 | 1 | 001 | 0 | 0 | 1 | 0 | 1 | 1 |
| AND Wb, Ws, Wd | 01100 | 1 | 0 | 1 | 0 | 010 | 0 | 0 | 1 | 0 | 1 | 1 |
| IOR Wb, Ws, Wd | 01110 | 1 | 0 | 1 | 0 | 011 | 0 | 0 | 1 | 0 | 1 | 1 |
| MOV f, Wnd | 10000 | 0 | 0 | 0 | 0 | - | 0 | 1 | 1 | 0 | 0 | 1 |
| MOV Wns, f | 10001 | 0 | 0 | 0 | 0 | - | 1 | 0 | 0 | 0 | 1 | 1 |
| BRA expr | 00110 | 0 | 0 | 0 | 0 | - | 0 | 0 | 0 | 1 | 1 | 1 |
| BRA C, expr | 00110 | 0 | 0 | 0 | 0 | - | 0 | 0 | 0 | 1 | 1 | 1 |
| BRA OV, expr | 00110 | 0 | 0 | 0 | 0 | - | 0 | 0 | 0 | 1 | 1 | 1 |
| BRA Z, expr | 00110 | 0 | 0 | 0 | 0 | - | 0 | 0 | 0 | 1 | 1 | 1 |
| BRA N, expr | 00110 | 0 | 0 | 0 | 0 | - | 0 | 0 | 0 | 1 | 1 | 1 |
| LSR Wb, Wns, Wnd | 11011 | 1 | 0 | 1 | 0 | 101 | 0 | 0 | 1 | 0 | 1 | 0 |
| NEG Ws, Wd | 11101 | 1 | 1 | 1 | 1 | 100 | 0 | 0 | 1 | 0 | 1 | 1 |
| BTG Ws, #bit4 | 10100 | 0 | 0 | 0 | 0 | 110 | 0 | 0 | 1 | 0 | 0 | 1 |
| SUBB Wb, #lit5, Wd | 01011 | 1 | 1 | 1 | 1 | 111 | 0 | 0 | 1 | 0 | 1 | 1 |