Student: Frasineanu Claudia Andreea
Group: CEN 3.2A
Project number: 31

PIC24 Documentation

In the table below are the specific instructions corresponding to my project's number:

Non-jump Instructions	Flags	Jump Instructions
LSR Wb, #lit4, Wnd	OV	BRA OV, Expr
ZE Ws, Wnd	C	BRA C, Expr
BTST.Z Ws, #bit4	N	BRA N, Expr
INC Ws, Wd	Z	BRA Z, Expr

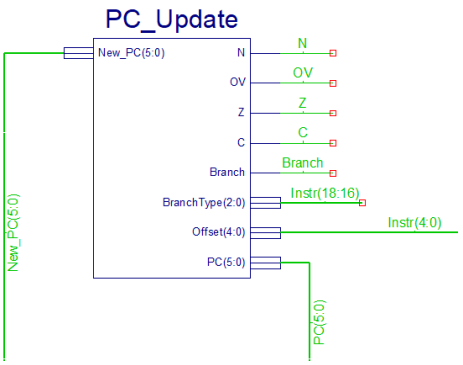
The microprocessor is composed of a total of 9 individual elements, each having an important and crucial role. Starting from the upper leftmost element, we have the following blocks:

I. PC_Update

This block is responsible for computing the new value of the *Program Counter (PC)*. Depending on the type of the instruction, the new value of the *PC* will either be:

- *PC* + 2, for a normal instruction;
- *PC* + 2 + *offset**2, for a branch instruction.

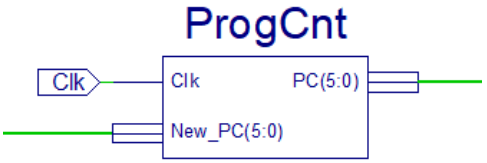
In the last case, a left shift is performed on the offset before it is added. The offset can be parsed from bits 15 → 0, but we can also use just the last 5 bits.



It should be noted that the new value of the *PC* is only an intermediar value, and will not be attributed to the *PC* through this block.

II. Program Counter

This is the block responsible for attributing the newly determined value of the *PC*. The block takes the value computed in the previous block (PC_Update) and attributes it to the *PC*.



As a remark, this operation will only be performed on a rising edge of the Clock (or when the signal changes from 0 to 1).

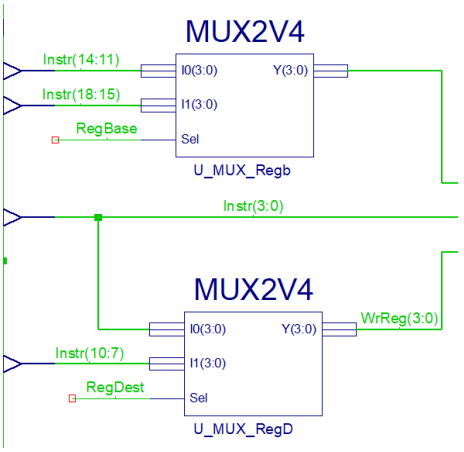
III. ROM32x24

This block represents the memory of the microprocessor. It contains 32 words, each word having 24 bits.

Using the first 5 bits of the *PC*, the corresponding word will be chosen. This represents a 24 bits instruction, which will be used in the other blocks.

IV. MUX2V4

These are simple multiplexores, used to decide which interval of the instruction needs to be used for this instance. The upper one is used for the *base register*, whilst the lower one is used for the *destination register*.



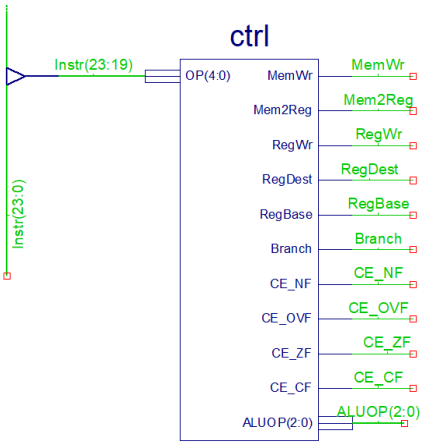
From the first table (completed in Milestone 1) we can see that the *destination registers* are either between the bits 10-7 or 3-0.

The *base registers* are either between the bits 18-15 or 14-11.

For the first MUX, when *RegBase* is 0, the value entering through *I0* is selected, and when it is 1, the value of *I1* is selected. The output is represented by the selected value.

V. Ctrl

This block is responsible for setting the different types of signals in conformity with the *OPCODE*. The values are updated in accordance with the second table (created for Milestone 2).



Starting with the flags, we have:

- *CE_NF* = 1 (negative flag is active)
- *CE_OVF* = 1 (overflow flag is active)
- *CE_ZF* = 1 (zero flag is active)
- *CE_CF* = 1 (carry flag is active)

All of these flags are sent to ALU so that it can correctly compute the operations.

Also for ALU, we have the signal *ALUOP*, a 3 bits number, used to decide the current instruction.

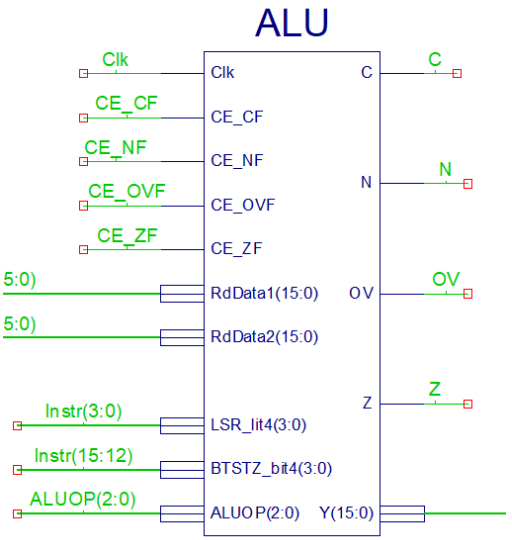
Starting from the top towards the bottom, we have the following signals:

- *MemWr*
  - signal used in *DataMem* block to enable the memory writing (it is enabled when the signal is 1);
  - the only instruction writing in memory is *MOV Wns, f*.
- *Mem2Reg*
  - signal used in *MUX2V16* to select whether the information is taken from the block *DataMem* or *ALU*;
  - for 0 we read from *ALU* and for 1 we read from *DataMem*.

- *RegWr*
  - signal used in the block File_Regs to enable writing in the registers (it is enabled when the signal is 1).
- *Branch*
  - signal used in PC_Update block in order to enable a branch jump (it is enabled when the signal is 1).
- *RegBase*
  - signal used in the block MUX2V4 (U_MUX_Regb) to decide which input is chosen (the bits 18-15 or 14-11 of the instruction).
- *RegDest*
  - signal used in the block MUX2V4 (U_MUX_Regd) to decide which input is chosen (the bits 10-7 or 3-0 of the instruction).

VI. ALU

This block is responsible for the arithmetical and logical instructions. The central elements of this block are *ALUOP* (sent from the *ctrl* block to decide which operation is performed), *RdData1* (from the base register) and *RdData2* (from the source register), which are the operands.



Then we have the *Clock* signal and the flags sent from the *ctrl* block.

A. **LSR Wb, #lit4, Wnd** (Logical Right Shift by short literal). The value within the register *Wb* is right shifted by the number of bits specified by the literal value, and the result is stored in the *Wnd* register.

B. **ZE Ws, Wnd** (Zero extend). Zero-extends the least significant Byte from the register *Ws* and stores the result into the register *Wnd.*

Ws(7:0) → Wnd(7:0) and
0 → Wnd(15:8)
Ex: ZE W1, W2 where (W1) = 0x1234 ⇒ (W2) ← 0x0034

C. **BTST.Z Ws, #bit4** (Test in Ws). The number given through *#bit4* will represent the position of the bit which will be tested. Then, after testing the bit, it will be overwritten in *Z* the negated value of the tested bit.

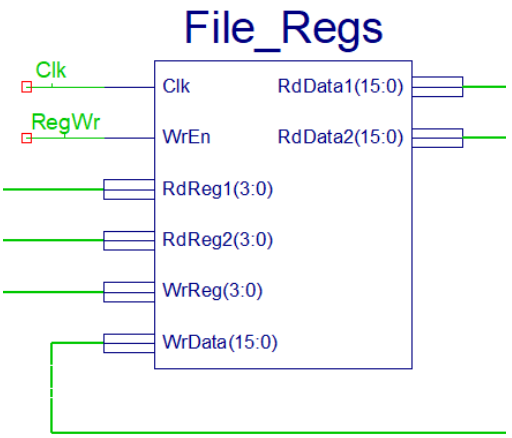
Ex: BTST.Z W1, 0x7 ⇒ Z = 1 (the seventh bit is 0)

D. **INC Ws, Wd** (Increment Ws). Adds 1 to the value inside the *Ws* register and stores the result inside the *Wd* register.

Ex: INC W1, W2 where (W1) = 0900 ⇒ (W2) ← 0911

VII. File_Regs

Inside this block we have 16 registers, noted from W0 to W15, each having a size of 16 bits. The registers are used to retain the data necessary for the operations. Thus, we write the data that has a destination register and read the data from the corresponding source register.

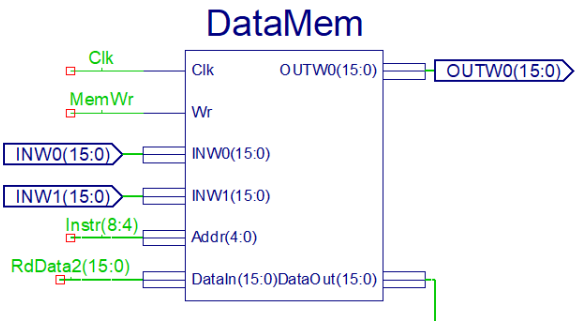


RdRed1 and *RdReg2* represent the addresses from which we want to take the values of the operands. The actual values are sent to the *RdData1* and *RdData2*. Likewise, *WrReg* holds the address at which we want to write the information, and *WrData* contains the actual value that we want to write.

It should be noted that the data is written only when the writing is enabled (*RegWr* is 1) and on the rising edge of the Clock signal.

VIII. DataMem

Inside this block there is a 16x16 RAM module (16 words, each having 16 bits), and is the place where the instruction *MOV Wns, f* writes.

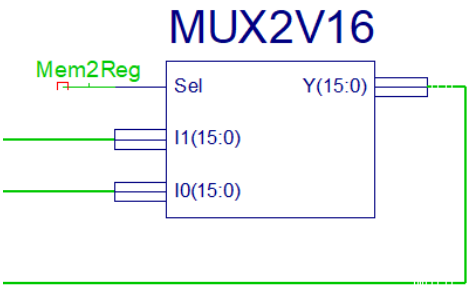


The input is read from the addresses 1020h (*INW0*) and 1022h (*INW1*), and the output is written at the address 1024h (*OUTW0*)

Based on the value of the *Addr*, the corresponding value inside the memory is selected and sent out through the signal *DataOut* to the *MUX2V16* block.

IX. MUX2V16

The purpose of this MUX is to decide whether the information is read from the *DataMem* block or *ALU*, depending on the value of the signal *Mem2Reg*, sent from the *ctrl* block.



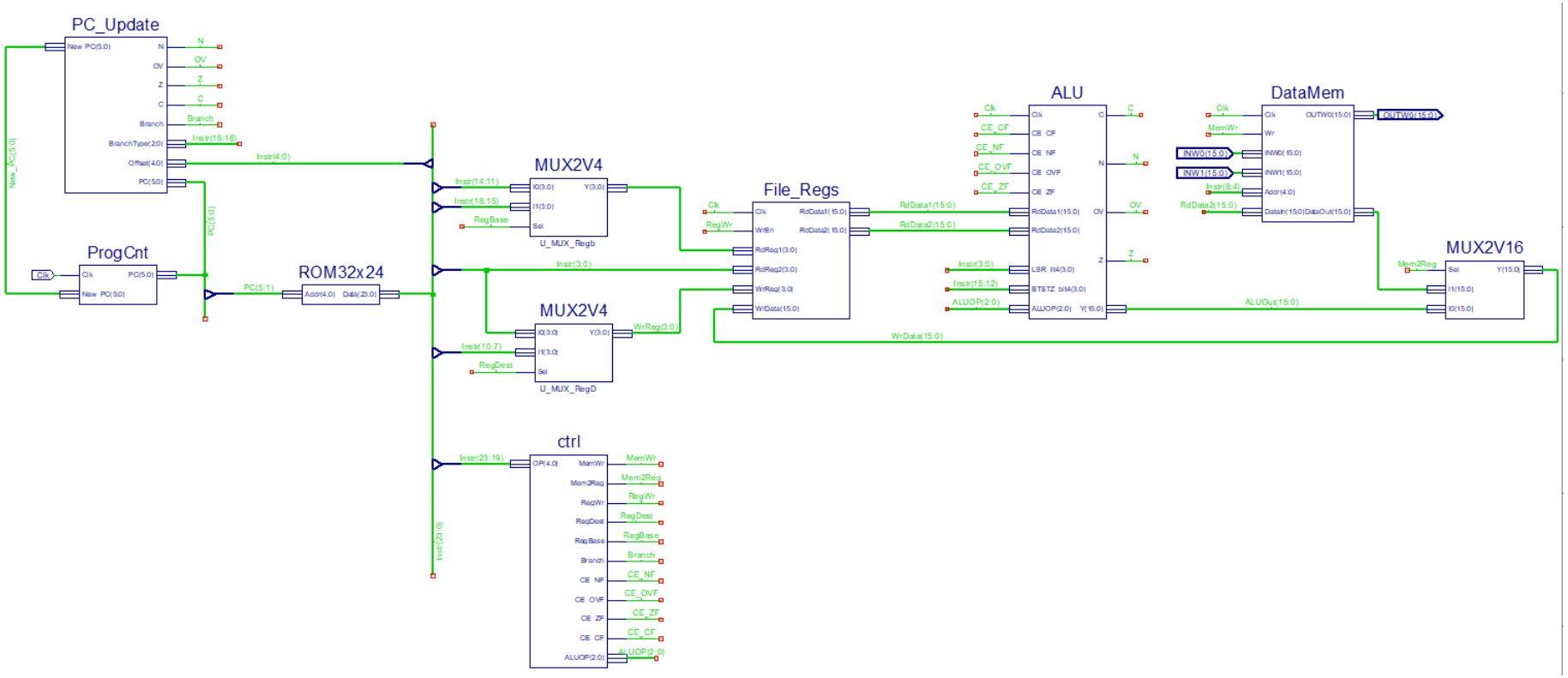
The corresponding value is then sent in the *File_Regs* block as the actual results of the operation.

Instruction coding (Milestone 1)

Encoding	2222 3210	1111 9876	1111 5432	11 1098	7654	321 0	Flags
ADD	0100	0www	wBqq	qddd	dppp	ssss	N, OV, Z, C
SUB	0101	0www	wBqq	qddd	dppp	ssss	N, OV, Z, C
AND	0110	0www	wBqq	qddd	dppp	ssss	N, - , Z, -
IOR	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
LSR Wb, #lit4, Wnd	1101	1110	0www	wddd	d100	kkkk	N, -, Z, -
ZE Ws, Wnd	1111	1011	1000	0ddd	dppp	ssss	N, -, Z, C
BTST .Z Ws, #bit4	1010	0011	bbbb	z000 0	0ppp	ssss	-, -, Z, C
INC Ws, Wd	1110	1000	0Bqq	qddd	dppp	ssss	DC, N, OV, Z, C
BRA OV, Expr	0011	0000	nnnn	nnnn	nnnn	nnnn	none
BRA C, Expr	0011	0001	nnnn	nnnn	nnnn	nnnn	none
BRA N, Expr	0011	0011	nnnn	nnnn	nnnn	nnnn	none
BRA Z, Expr	0011	0010	nnnn	nnnn	nnnn	nnnn	None

Ctrl signals (Milestone 2):

Encoding	OPCODE	CE_NF	CE_OVF	CE_ZF	CE_CF	ALUOP	MemWr	Mem2Reg	RegWr	Branch	RegDest	RegBase
ADD	01000	1	1	1	1	000	x	x	1	x	1	1
SUB	01010	1	1	1	1	001	x	x	1	x	1	1
AND	01100	1	x	1	x	010	x	x	1	x	1	1
IOR	01110	1	x	1	x	011	x	x	1	x	1	1
MOV f, wnd	10000	x	x	x	x	-	x	1	1	x	0	x
MOV wns, f	10001	x	x	x	x	-	1	x	x	x	x	x
BRA expr	00110	x	x	x	x	-	x	x	x	1	x	x
LSR Wb, #lit4, Wnd	11011	1	x	1	x	100	x	x	1	x	1	0
ZE Ws, Wnd	11111	1	x	1	1	101	x	x	1	x	1	x
BTST .Z Ws, #bit4	10100	x	x	1	1	110	x	x	1	x	x	x
INC Ws, Wd	11101	1	1	1	1	111	x	x	1	x	1	x
BRA OV, Expr	00110	x	x	x	x	-	x	x	x	1	x	x
BRA C, Expr	00110	x	x	x	x	-	x	x	x	1	x	x
BRA N, Expr	00110	x	x	x	x	-	x	x	x	1	x	x
BRA Z, Expr	00110	x	x	x	x	-	x	x	x	1	x	x


PC_Update
New PC(5:0)
N
OV
Z
C
Branch
BranchType(2:0)
Offset(4:0)
PC(5:0)
Instr(18:16)
Instr(4:0)
New_PC(5:0)
PC(5:0)
ProgCnt
Clk
PC(5:0)
New PC(5:0)
PC(5:1)
ROM32x24
Addr(4:0)
Data(23:0)
MUX2V4
I0(3:0)
I1(3:0)
Sel
Y(3:0)
Instr(14:11)
Instr(18:15)
RegBase
U_MUX_Regb
Instr(3:0)
MUX2V4
WrReg(3:0)
Instr(10:7)
RegDest
U_MUX_RegD
ctrl
OP(4:0)
Instr(23:19)
Instr(23:0)
MemWr
Mem2Reg
RegWr
RegDest
RegBase
Branch
CE_NF
CE_OVF
CE_ZF
CE_CF
ALUOP(2:0)
File_Regs
Clk
WrEn
RegWr
RdReg1(3:0)
RdReg2(3:0)
WrReg(3:0)
WrData(15:0)
RdData1(15:0)
RdData2(15:0)
ALU
Clk
CE_CF
CE_NF
CE_OVF
CE_ZF
RdData1(15:0)
RdData2(15:0)
Instr(3:0)
Instr(15:12)
ALUOP(2:0)
LSR II4(3:0)
BTSTZ bit4(3:0)
ALUOP(2:0)
C
N
OV
Z
Y(15:0)
ALUOut(15:0)
WrData(15:0)
DataMem
Clk
MemWr
INW0(15:0)
INW1(15:0)
Instr(8:4)
RdData2(15:0)
Wr
INW0(15:0)
INW1(15:0)
Addr(4:0)
DataIn(15:0)
OUTW0(15:0)
DataOut(15:0)
OUTW0(15:0)
MUX2V16
Mem2Reg
Sel
I1(15:0)
I0(15:0)
Y(15:0)