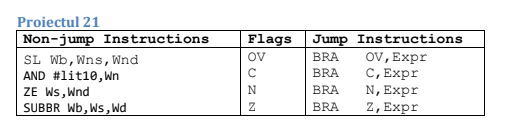
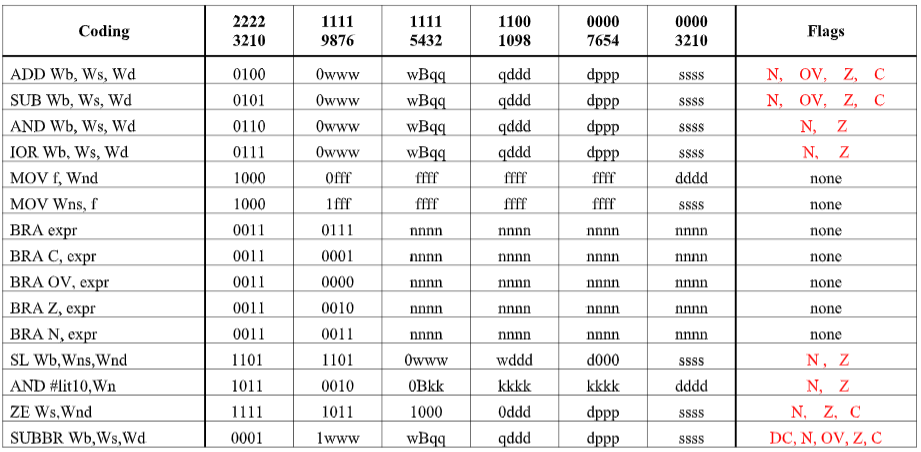
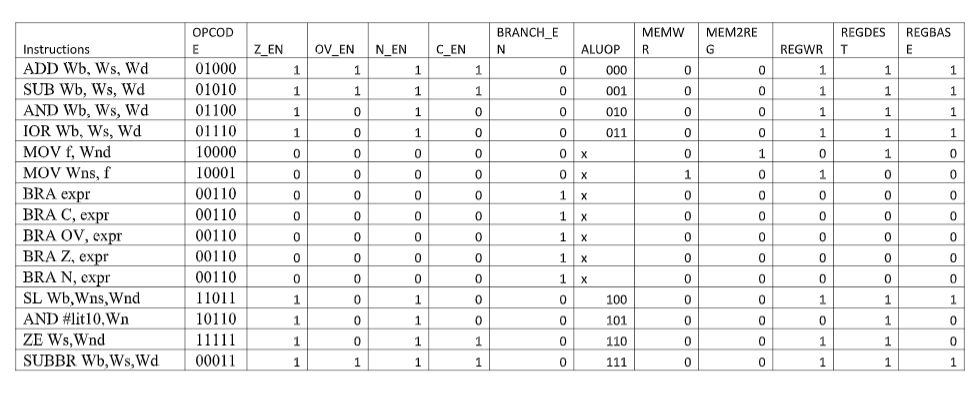
**Documentatie PIC24**

****

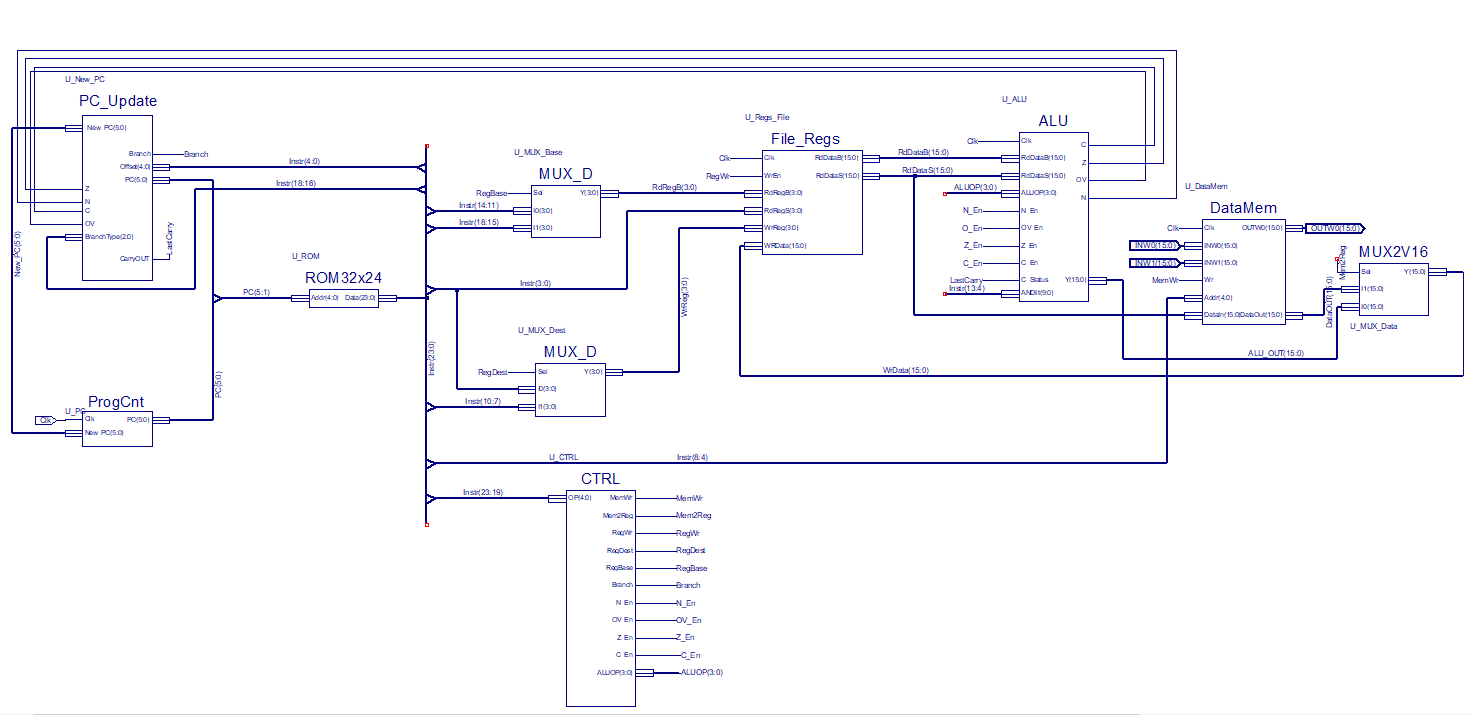
**Tabel 1-tabelul pentru instructiuni:**



**Tabel 2-tabelul pentru blocul de control:**



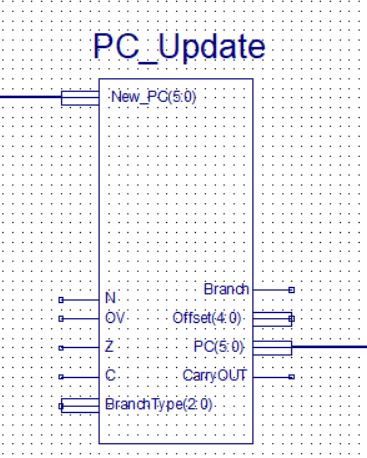
**Schema completa:**

****

**Descriere blocuri utilizate:**

**PC\_Update:**

Acest bloc actualizeaza contorul programului. La PIC24 cuvantul instructiune are lungimea de 2 octeti, PC va avea valorile 0,2,4…Aceste valori se obtin prin operatia PC=PC+2. Daca PC=000000, PC+2 va fi 000010. Offsetul este extins ca semn si deplasat stanga cu doi. Noua valoare va fi valoarea curenta plus deplasamentul la stanga cand Branch este 1 si unul dintre flag-uri este activ. Adica, daca exista o intstructiune de tip Branch sau alte conditii specificate, PC va primi PC+2+offset\*2. Offset-ul fiind luat din bitii Branch. Daca nu exista conditii specificate, atunci PC va fi incrementat cu 2.

****

N – flag-ul de negative

Z - flag-ul de zero

C - flag-ul de transport

OV - flag-ul de overflow

BranchType - semnalul pentru instructiunile de branch pentru a face distinctia intre cele 4 BRA

Branch - flag-ul de Branch

Offset - valoarea deplasamentului

PC - Program Counter-ul curent

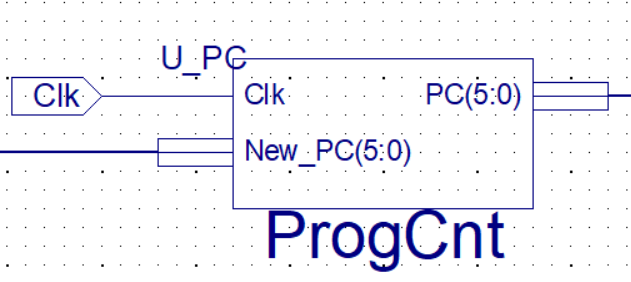
New\_PC - Noul Program Counter(calculat)

CarryOut - rezultatul transportului

**ProgCnt:**

Este utilizat pentru a actualiza contorul de program pe un ceas cu front ascendent. Mai exact, PC memoreaza valoarea lui New\_PC pe frontal ridicator al lui Clk. La fiecare impuls de tact, ProgCnt este actualizat de catre PC\_Update. PC\_Update primeste de la unitatea de control semnalul Bra ce semnifica executia unei instructiuni de Branch.

Valoarea initiala a lui PC este ‘0’ pentru toate elementele.

****

Clk - semnalul de Clock

New\_PC - noul Program Counter

PC - Program Counter-ul curent

**ROM32x24:**

Acest bloc este memoria efectiva a programului. Are 32 de cuvinte pentru instructiuni, fiecare de 24 biti. Memoria ROM contine instructiunile ce sunt executate de procesor. Instructiunea de iesire este selectata de valoarea transmisa de 4 biti ai contorului programului (5:1) inserati in bloc ca Addr. Addr reprezinta semnalul pentru adresa, iar Data reprezinta semnalul pentru date.

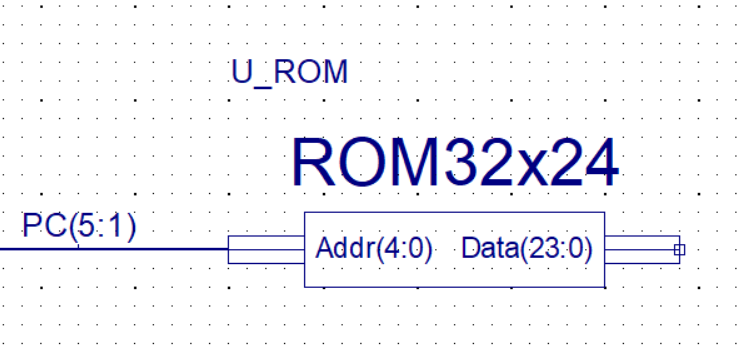
Deoarece se afla 32 de locatii de memorie, sunt necesari 5 biti pentru adresarea memoriei(toate valorile de 5 biti rezulta 32 de combinatii).

Addr ramane (4 downto 0) adica 5 biti pt ca este folosit sa parcurga linile lui tRom care are

32 de linii iar Data se face (23 downto 0) pt ca elementele lui tRom au 24 de biti.

Pentru a adresa 32 de locatii este nevoie de 5 terminale de adresa. Continutul locatiei de la adresa specificata prin intermediul terminalelor de adresa devine disponibila catre alte module prin intermediul a 24 de terminale de date.

In cazul citirii orice memorie genereaza la iesire continutul locatiei selectata de adresa. Elementul cel mai din dreapta al vectorului este considerat LSB iar mai din stanga este MSB.

****

Addr(4:0) - adresa de unde selectam instructiunea

Data(23:0) - iesirea blocului/instructiunea

**MUX2V4:**

Aceste multiplexoare sunt utilizate pentru selectarea registrelor de baza si de destinatie.

Intrarile blocului sunt selectate de variabila RegDest care este setata in blocul de control pentru fiecare instructiune, deoarece bitii destinatie pot fi gasiti in 2 pozitii/locatii diferite, intre bitii 10:7 si 3:0. Iar registrele de baza pot fi gasite intre bitii 18:15 si 14:11.

Conform tabelului cu instructiuni, se pot observa de exemplu la instructiunea ADD Wb, Ws, Wd ca bitii pentru adresarea registrului destinatie (Wd) sunt de la 10:7. Insa pentru instructiunea MOV f, Wnd se observa ca bitii pentru adresarea registrului destinatie(Wnd) sunt de la 3:0.

Din aceasta cauza, este necesar sa punem un multiplexor care sa selecteze bitii necesari pentru adresarea filei de registrii. De asemenea, Pentru instructiunea ADD Wb, Ws, Wd, se observa ca bitii pentru adresarea registrului baza (Wb) se afla de la 18:15.

In concluzie, avem nevoie, in acest caz, de doua multiplexoare ce sa selecteze bitii necesari instructiunii.

**U\_MUX\_Base:**

I1(3:0) - bitii pentru instructiunile commune

I0(3:0) - bitii pentru instructiunea SL Wb, Wns, Wnd

Sel - bitul de selectie

Y(3:0) - iesirea

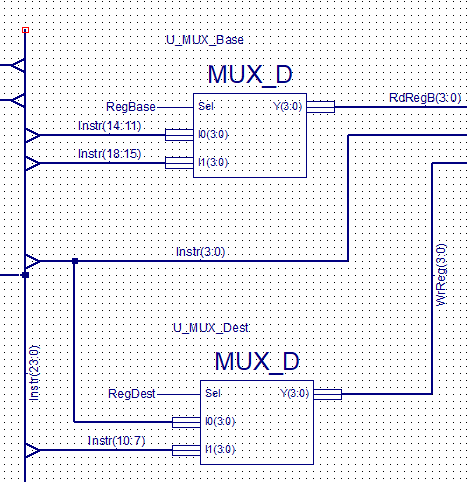
**U\_MUX\_Dest:**

I0(3:0) - bitii pentru instructiunea Mov f, Wnd si AND #lit10, Wn

I1(3:0) - bitii pentru celelalte

Sel-bitul de selectie

Y(3:0) - bitii folositi pentru selectarea registrului de destinatie



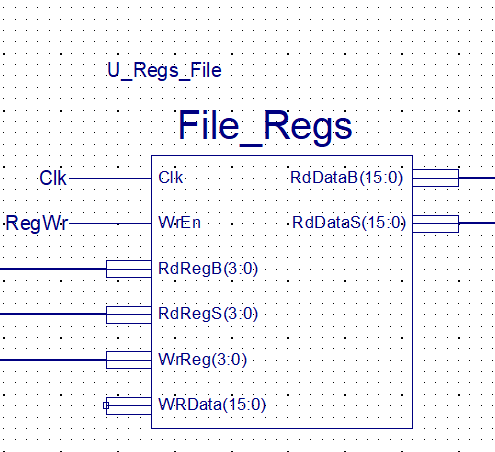
**FILE\_REGS:**

Acest bloc contine 16 registre (W0-W15) fiecare cu dimensiunea de 16 biti.

Aici scriem datele din operatiile care au registru de destinatie si citim informatiile pentru cele cu registru de baza sau sursa.

Fila de registre se actualizeaza la fiecare semnal de tact pe front ridicator, si cu semnalul RegWr activ, ce vine de la unitatea de control.

Instructiunea MOV f, Wnd muta informatia din memoria RAM in registrul destinatie specificat.

****

Clk - semnalul de clock

WrEn - semnal care activeaza scrierea

RdRegB(3:0) - bitii de adresa pentru adresa registrului de baza

RdRegS(3:0) - bitii de adresa pentru adresa registrului de sursa

WrReg(3:0) - bitii de adresa pentru adresa registrului destinatie

WrData(15:0) - data pentru scriere

RdDataB(15:0) - data din registrul de baza

RdDataS(15:0) - data din registrul sursa

**ALU:**

In acest bloc sunt calculate instructiunile precum si flagurile rezultate din calcul. Rezultatul este calculate, pe baza ALUOP din blocul CTRL, folosind RdDataB(registrul de baza) si RdDataS(registrul sursa). ALU are incorporat registrul de stare ce cuprinde flag-urile, de aceea ALU are ca si semnal de intrare Clk(tact). ALU primeste semnalul ALUOP ce reprezinta tipul de instructiune ce trebuie executat si semnalele de enable pentru flag-uri.

RdDataB(15:0) - operand de intrare (registrul de baza)

RdDataS(15:0) – operand de intrare (registrul sursa)

ALUOP(3:0) - operatia realizata de ALU

CLK – semnalul de Clk

N\_En – semnalul care indica daca putem modifica flagul N

Z\_En – semnalul care indica daca putem modifica flagul Z

C\_En – semnalul care indica daca putem modifica flagul C

OV\_En – semnalul care indica daca putem modifica flagul OV

ANDlit (9:0) – operand de intrare, contine bitii de instructiune 13:4 pe care se afla combinatia ce constituie litera necesara pentru instructiunea AND#lit10, Wn

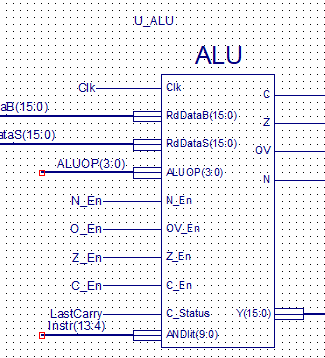
C - flag-ul C

Z - flag-ul Z

OV - flag-ul OV

N - flag-ul N

Y(15:0) – operand de iesire

****

Instructiunile commune implica operatii simple care se realizeaza cu ajutorul operatorilor +, -, AND si OR din compilator**.**

**Instructiunile individuale:**

**SL Wb, Wns, Wnd**

Wnd = left shift Wb by Wns, MSb → C

Deplaseaza la stanga continutul registrului sursa Wb cu cei mai putin semnificativi 5 biti ai Wns (doar pana la 15 pozitii) si stocheaza rezultatul in registrul destinatie Wnd. Orice biti mutate din registrul sursa sunt pierduti. Pentru Wb, Wns si Wnd trebuie utilizate adresarea directa de inregistrare.

**AND #lit10, Wn**

Wn = lit10 .AND. Wn (lit10=10-bit literal (0:255 in Byte mode, 0:1023 in Word mode)).

Calculam operatia logica AND a operandului literal de 10 biti si a continutului registrului de lucru Wn si plaseaza rezultatul inapoi in registrul de lucru Wn. Trebuie sa utilizam adresarea directa pentru Wn.

**ZE Ws, Wnd**

Wnd = zero-extended Ws

Extinde cu zero octetul cel mai putin semnificativ din registrul de lucru sursa Ws la o valoare de 16 biti si stocheaza rezultatul in registrul de lucru destinatie Wnd. Pentru registrul Ws poate fi utilizate adresarea directa sau indirecta, iar pentru registrul Wnd trebuie utilizate adresarea directa. Flag-ul N este sters, iar indicatorul C este setat, deocarece cuvantul extins cu zero este intotdeauna pozitiv.

**SUBBR Wb, Ws, Wd**

Wd = Ws – Wb – (C)

Scade continutul registrului de baza Wb si al inversului flagului Carry din continutul registrului sursa Ws si pune rezultatul in registrul de destinatie Wd. Pentru registrul Wb se utilizeaza adresarea directa, iar pentru registrele Ws si Wd se poate utiliza adresarea directa sau indirecta.

**DataMem:**

Acest bloc contine un modul RAM16x16.Acesta este locul unde scrie instructiunea MOV Wns, f. MOV Wns, f muta informatia din registrul sursa in memoria RAM.

Citirea se face de la adresele 1020h (INW0) si 1022h (INW1) iar scrierea se face la adresa 1024 (OUTW0) . Iesirea DataOut va avea informatiile din MemData, care este o valoare selectata din memorie pe baza valorii Addr.

Clk- semnalul de clock

INW0(15:0) – adresa de citire

INW1(15:0) – adresa de citire

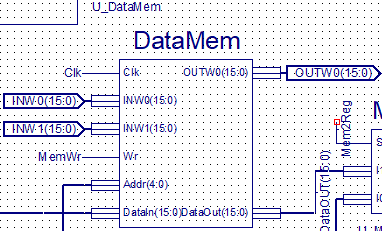
Wr – semnalul care indica scrierea in memorie

Addr(4:0) – bitii care indica adresa

DataIn(15:0) – valoarea care trebuie scrisa

DataOut(15:0) – primeste INW0 sau INW1

OUTW(15:0) – adresa de scriere

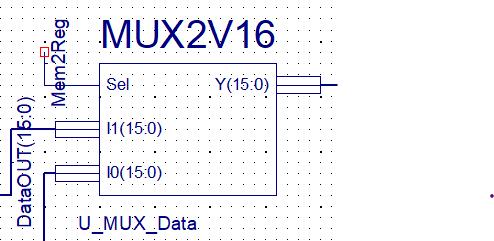
****

**MUX2V16:**

Acest bloc decide pe baza semnalului Mem2Reg, daca datele sunt scrise in registru din blocul ALU sau din memorie (MOW f, Wnd). Multiplexorul are 2 intrari a cate 16 biti, si se ocupa cu stabilirea caii catre FileRegs.

De exemplu, daca avem instructiunea ADD Wb, Ws, Wd, va fi executata de ALU si rezultatul trebuie stocat in registrul destinatie Wd. Astfel, multiplexorul trebuie sa permita calea rezultatului de la ALU catre FileRegs.

Pentru asta, unitatea de control ofera semnalul Mem2Reg = 0. Insa, daca se executa instructiunea MOV f, Wnd data din DataMem trebuie scrisa in registrul destinatie iar multiplexorul trebuie sa selecteze calea de la DataMem la FileRegs.

****

Sel – selectia, daca este 0 se selecteaza data provenita din ALU

I0(15:0) – data din ALU

I1(15:0) – data din memorie

Y(15:0) - iesirea

**CTRL:**

Acest bloc trimite toate valorile utilizate de celelalte blocuri. Aceste valori sunt selectate de OPCODE-ul intructiunii primite. Unitatea de control decodifica instructiunea si da informatiile necesare ca de exemplu enable-urile pt registrii si memorie, selectoarele pentru multiplexoare, ALUOP pentru ALU, enable-urile pt flaguri etc.

**Semnale:**

**MemWr**

* utilizat pentru scrierea in memorie;
* singura instructiune care scrie in memorie este MOV Wns, f;

**Mem2Reg**

* daca semnalul este activat atunci datele sunt scrise in registrul memorie, altfel datele scrise vor fi iesirea ALU;
* data va fi scrisa din memorie in registru;

**ALUOP**

* reprezinta id-ul de selectie al instructiunii curente;
* determina operatia in ALU;

**RegWr**

* utlizat in File\_Regs pentru a permite scrierea in register;
* 1 da, 0 nu;

**RegBase**

* semnal utilizat in Mux pentru selectarea bitilor de baza;
* selectarea se face din bitii 18:15 si 14:11;
* daca acest semnal este 0, **U\_MUX\_Base** va alege intrarea I0**,** altfel va alege I1 .

**RegDest**

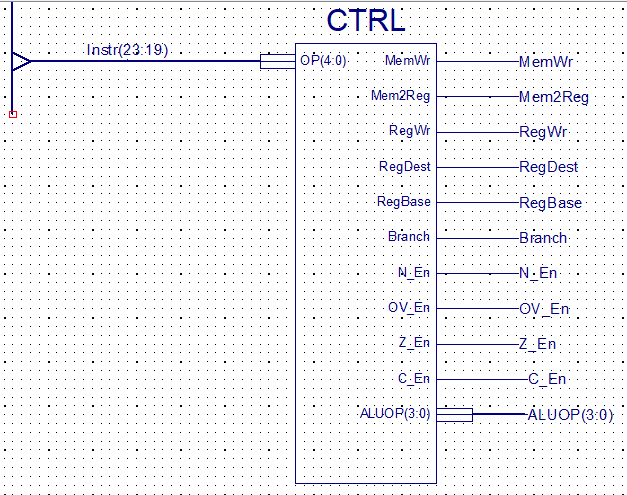
* semnal utilizat in Mux pentru selectarea bitilor de destinatie;
* selectarea se face din bitii 10:7 si 3:0;
* daca acest semnal este 0, **U\_MUX\_Dest** va alege intrarea I0, altfel va alege I1;

**Branch**

* semnal utilizat in PC\_Update;
* este active atunci cand avem o instructiune de Branch. Mai avem si BranchType care extrage instr(18:16) si indica ce tip de branch trebuie sa efectuam.

**OP**

* Opcode-ul intructiunuii curente;

****

De exemplu, pentru instructiunea ADD Wb, Ws, Wd toate flagurile vor fi active, deoarece aceasta instructiune influenteaza valoarea fiecarui flag.

Mai exact, in momentul in care vom executa aceasta instructiune, valorile flagurilor se vor schimba in functie de rezultatul apelului.