20…..év …hó …nap

NÉV:……**MINTA ZH2**..... Neptun kód:..…….. gyak/lab kurzus: .....….

A feladatokat önállóan, meg nem engedett segédeszközök használata nélkül oldottam meg:

Olvasható aláírás:.....……

Kedves Kolléga! ***A kitöltést a dátum, név és aláírás rovatokkal kezdje!*** Az alábbi kérdésekre a válaszokat - ahol lehet - mindig a feladatlapon oldja meg! A feladatok megoldása során a részletes kidolgozást nagyfeladatonként külön papíron végezze, (egyértelműen jelölje, hogy melyik lap melyik feladathoz tartozik, a papírra már a kezdetkor írja rá a nevét és Neptun kódját) és ezeket a papírokat is adja be a dolgozatával! A kérdésekre a táblázatok vagy a pontozott vonalak értelemszerű kitöltésével válaszoljon, hacsak külön másként nem kérjük. ***Mindenütt a legegyszerűbb megoldás éri a legtöbb pontot.*** Ha több a feladat, mint a pont, a feladat pontszámát csökkentjük minden hiba esetén (azonban negatív pontot nem adunk). Jó munkát!

E: (25p)
F1: (15p)
F2: (25p)
$\sum$: (65p)
IMSC: (7p)

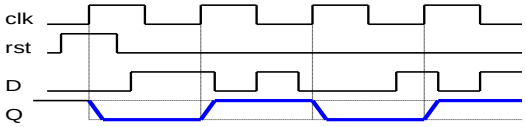
Ellenőrző kérdések (25p)

E1. (4p) **a.** Egy felfutó él érzékeny szinkron törölhető (rst) D flip-flop az alábbi jeleket kapja. Rajzolja le a Q kimenet idődiagramját! **b.** Adja meg egy törlő (rst) bemenettel rendelkező D flip-flop Verilog viselkedési leírását! Elkezdtük, folytassa!

always@(**posedge clk**)

if(**rst**) .....**Q <= 1'b0;**.....

..**else**.... .... **Q <= D;**.....



E2. (3p) Adja meg egy 2 db 8 bites forrásból betölteni képes ***multifunkciós regiszter*** Verilog leírását! ***Ha ld0=1, q-ba töltse be In0-at, ha ld1=1, q-ba töltse be In1-et! Az ld0 legyen nagyobb prioritású!*** (Most nem kell alaphelyzetbe állítás rst-re.) Elkezdtük, folytassa!

A hiányzó változó definícióktól eltekintünk.

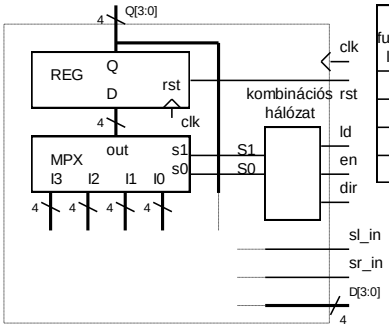
always@(**posedge clk**)

if(...**ld0**.....) .....**q <= In0;**.....

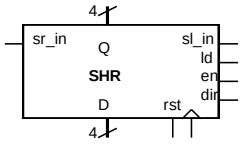
..**else**.. ..**if(ld1)**.. .... **q <= In1;**.....

E3. (4p) Adott egy 2 irányú shiftregiszter funkcionális blokkvázlata és működési táblázata.

a. Adja meg, bitsorrend helyesen, hogy milyen jelek kapcsolódnak az MPX multiplexer egyes bementeire! Ügyeljen arra, hogy a rajzon látható jel neveket használja!



dekódolt funkcióválasztás					
ld	en	dir	S1	S0	funkció
0	0	x	0	0	tart
0	1	1	1	0	jobbra shift
0	1	..0	1	1	balra shift
..1	..x	..x	0	1	tölt



I0:.....**Q[3:0]**.....

I1:.....**D[3:0]**.....

I2:.....**sr_in, Q[3:1]**.....

I3:.....**Q[2:0], sl_in**.....

b. Töltse ki a táblázat hiányzó részeit, ha a prioritás sorrendje a legerősebb jellel kezdve a következő: betöltés (ld), engedélyezés (en)!

E4. (5p) Adja meg egy 4 bites 12-es modulusú fel le számláló Verilog kódját! A számláló ld-ra betölti d[3:0]-at, en=1 esetén számol, dir=1 esetén felfele, dir=0 esetén lefele, egyébként nem változik a q kimenete. A vezérlőjelek prioritása a legerősebb jellel kezdve a következő: ld, en. A jelek definiálásától most eltekintünk.

assign tc = **en&(dir&(q==4'd11) | ~dir&(q==4'b0));**
always@(**posedge clk**)

if(...**ld**....) q <= .....**d;**.....

else if(.....**en**...)

if(...**dir**...)

if(....**tc**.....) q <= **4'd0;**.....
vagy if(q==4'd11)

else..... **q <= q + 4'd1;**.....

else

if(**tc**)... q <= **4'd11;**.....
vagy if(q==4'b0)

else…. **q <= q - 4'd1;**.....

E5. (3p) Adott 3 db számláló modul példányosítva, azonban az összekötésük hiányzik. Az első 7-es modulusú, a második 3-os modulusú, a harmadik 5-es modulusú. A számláló kimenetek definiálásától eltekintünk.
a. Kaszkádosítsa a felsorolás szerinti sorrendben őket megadott wire típusú jelek megfelelő felhasználásával úgy, hogy a teljes számláncot a külső ***EN*** jel engedélyezze!

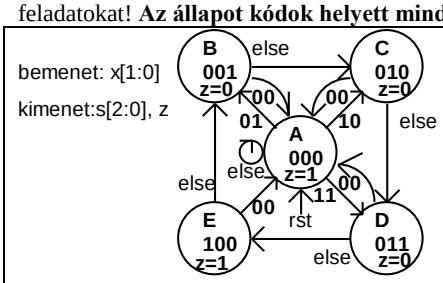
```
wire EN, OUT;
wire [1:0] tc_i;

cnt7 cnt7_1(.clk(..clk...), .rst(..rst....),.en(...EN...),   .q(q7), .tc(...tc_i[0]....));

cnt3 cnt3_1(.clk(..clk...), .rst(..rst....),.en(..tc_i[0].),  .q(q3), .tc(...tc_i[1]....));

cnt5 cnt5_1(.clk(..clk....), .rst(..rst....),.en(..tc_i[1].), .q(q5), .tc(OUT));
```

b. Hány órajelenként jelenik meg impulzus az ***OUT*** kimeneten? ....**7*3*5=105**.....

E6. (6p) Mely állítások igazak és melyek hamisak? Jelölje **+*-al az igaz, -*-al a hamis** állításokat!

1.	Egy sorrendi hálózat (FSM) kódolt állapottáblája tartalmazza a next_state logika és a kimeneti logika igazságtábláit.	+
2.	Ha egy sorrendi hálózat kimenetét az állapotregszter valamely bitjei adják, akkor az biztosan Moore modell szerint működik.	+
3.	A **FIFO**-ba bármikor lehet adatot írni.	-
4.	Egy FSM vezérlő állapotgráfjába írt állapotátmeneti feltétel lehet A == B.	-
5.	Az **ASM** (algoritmikus állapotvezérlő) leírás egyetlen feltétel doboza csak 2-felé ágazást tesz lehetővé.	+
6.	3 regiszter címes processzor architektúra esetén egy művelet eredménye nem rontja el szükségszerűen az egyik operandust.	+

F1. (15p) Adott egy FSM az alábbi ***kódolt állapotgráfjával.*** A kimenete az állapotkód és **z**. Végezze el az alábbi feladatokat! **Az állapot kódok helyett mindenütt állapot neveket használjon!**



a. (3p) Adja meg az állapotregiszter Verilog leírását!

```
//A konstans és változó deklarációk
localparam [2:0] A = 3'b000, B = 3'b001,
    C = 3'b010, D = 3'b011, E = 3'b100;
reg [2:0] s, next_s;
wire clk, rst;
wire [1:0] x;
//Állapotregiszter
always @ ( …posedge clk…)

   if (rst)  s <= …A…;

   else      s <= …next_s…;
```

b. (9p) Következő állapot logika

```
//Next_state logika
always @ (.......*.......)
   case (s)
    A:  case(…x…)
        2'b01:   next_s <= ……B……;

        2'b10:   next_s <= ……C……;

        2'b11:   next_s <= ……D……;

        default: next_s <= ……A……;
        endcase
    B: if(x == 2'b00) next_s <= ……A……;

       else           next_s <= C;

    C: if(x == 2'b00) next_s <= ……A……;

       else           next_s <= ……D……;

    D: if(x == 2'b00) next_s <= ……A……;

       else           next_s <= ……E……;

    E: if(x == 2'b00) next_s <= ……A……;

       else           next_s <= ……B……;

default:              next_s <= ……A……;
    endcase
```

c. (1p) Milyen modell szerint működik? (Húzza alá a megfelelőt) Mealy Moore

d. (2p) Adja meg a z kimenet logikai függvényét Verilog-ban!

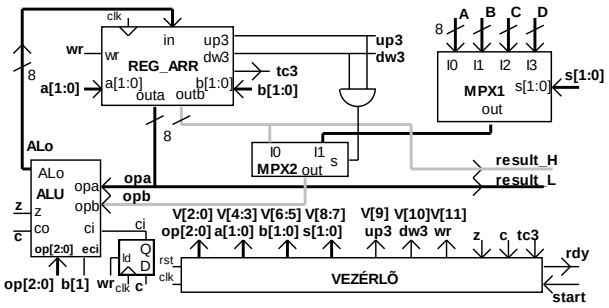
Ha csak az s-t, az állapot neveket, az == operátort és | műveletet használhatja:

assign z = …**(s == A)|(s == E);**…

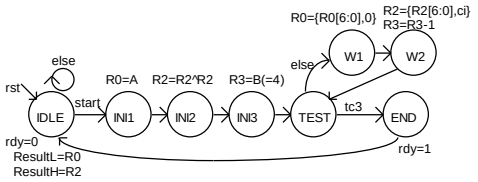
A legegyszerűbb Boole algebrai alakban: **assign z =** ……**~s[1]&~s[0];**…

F2. (25p) Funkcionális blokkvázlatával (adatstruktúra + vezérlő) adott egy a bemenő adatokkal többféle művelet elvégzésére alkalmas számító modul. Az adatstruktúra **4 db 8 bites külső adat bemenettel** rendelkezik: **A, B, C, D.** Az aritmetikai logikai egység **(ALU**) 8 műveletet tud elvégezni az **opa** és **opb** operandusokkal az alább megadott táblázat szerint. Az **elvégzendő művelet** az ALU **op[2:0]** bemenetén állítható be. **Shiftelés esetén a belépő bit 0, ha eci = 0, ci, ha eci =1. Összeadás/kivonás esetén a ci-t csak akkor veszi figyelembe, ha eci = 1**. A **művelet eredményét** az **ALo** kimenet adja. Az ALU **z** kimenet 1 értéke jelzi, ha a **művelet eredménye 0**. A **c** kimenet **összeadás esetén a túlcsordulást, kivonás esetén a negatív eredményt, shiftelés esetén a kilépő bit** értékét jelzi. A **REG_ARR egy 4 regisztert tartalmazó regiszter tömb.** Ennek **outa kimenetén az a[1:0]-val kiválasztott sorszámú regiszter tartalma** jelenik meg, **ha a[1:0]=i, akkor Ri**. Az **adat beírását wr=1 engedélyezi és az ALU-ból az in bementére jövő adat** (ALo) **az a[1:0]-val kiválasztott regiszterbe** íródik az órajel felfutó élére. A **wr jel** ezen kívül **az ALU c kimenetét beírja az ALU ci bementére kapcsolt 1 bites**

regiszterbe. A regiszter tömb **outb kimenetén a b[1:0]-val kiválasztott regiszter tartalma** jelenik meg, **ha b[1:0]=i, akkor Ri**. Az ALU **eci** bemenetére **b[1]** van kötve, ezért a **ci bemenetét csak akkor veszi figyelembe az ALU, ha opb-n R2 vagy R3 van kiválasztva**. **Az ALU opb bemenetére kerülő adat** az MPX2 multiplexerről jön, mely **alapesetben (dw3,up3 != 11) a regiszter outb** kimenetét választja ki. **Ha dw3,up3=11,** akkor az **s[1:0]-al kiválasztott külső adat lesz az opb. A regisztertömb R3 regisztere speciális, mert fel/le számlálóként is funkcionál.** Normál regiszterként írható (a[1:0]=2'b11 és wr = 1 esetén beíródik R3-ba az Alo-ról jövő érték) és normál regiszterként olvasható**.** Azonban, ha éppen nem írják, akkor **dw3,up3 = 1,0 esetén lefele számol (R3=R3-1), dw3,up3 = 0,1 esetén pedig felfele számol (R3=R3+1),** ha megjön a clk aktív éle**. Az írás erősebb, mint a számlálás engedélyezés. Az R3 regiszter fel vagy le számláltatásával egyidőben bármely más regiszterbe lehetséges írni.** A **tc3** kimenet akkor jelez, ha **R3==0**. A vezérlőt egy kívülről jövő, 1 órajel hosszú **start** parancs indítja. A vezérlő az adatstruktúrát a **V[11:0] vezérlő jelekkel** működteti. A működése közben figyelni képes az adatstruktúrából jövő **z, c** és **tc3** feltétel jeleket. Egy számítási feladat elkészültét 1 órajel hosszú **rdy** státus jellel kell jeleznie. Az eredménynek meg kell maradnia a következő start jelig. **Az aktuális művelet szempontjából érdektelen vezérlő jelek értékét 0-nak kell választani**. Egy maximum 8 bites végeredményt a regiszter tömb **outa** kimenetén (**result_L**) kell megjeleníteni. Ha a végeredmény megjelenítéséhez 8-nál több bit kell, akkor a többi bitet az **outb** kimeneten (**result_H**) kell megjeleníteni.



a. (10p) Az alábbi Moore jellegű HLSM állapot diagram által leírt feladatot a fenti adatstruktúrával kell megvalósítani. (R0-at szorozza 16-al, eredmény:R2R0-ban)



ALU funkció vezérlés: op[2:0]	ALU out (Alo)	z =1, ha	c = 1, ha
000	opa + opb + ci&eci	Alo==0	átvitel van
001	opa – opb – ci&eci	Alo==0	a–b < 0
010	{opa[6:0], ci&eci}	Alo==0	opa[7]
011	{ci&eci, opa[7:1] }	Alo==0	opa[0]
100	opa & opb	Alo==0	Soha (c=0)
101	opa | opb	Alo==0	Soha (c=0)
110	opa ^ opb	Alo==0	Soha (c=0)
111	opb	Alo==0	Soha (c=0)

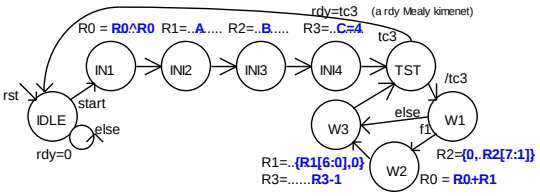
Adja meg a ***vezérlő állapotgráfjának megadott állapotaiban kiadandó vezérlőjeleket a V[11:0] bitek értékének (0, 1) alábbi táblázatba való beírásával!*** (Segítségképp néhány értéket megadtunk. A rdy-t most nem kérjük.)

	wr	dw3	up3	s[1:0]		b[1:0]		a[1:0]		op[2:0]		
V[11:0]	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
IDLE	**0**	**0**	**0**	0	0	**1**	**0**	**0**	**0**	0	0	0
INI1	**1**	**1**	**1**	**0**	**0**	0	0	**0**	**0**	**1**	**1**	**1**
INI2	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**
INI3	**1**	**1**	**1**	**0**	**1**	0	0	**1**	**1**	**1**	**1**	**1**
TEST	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0
W1	**1**	**0**	**0**	**0**	**0**	**0**	0	**0**	**0**	**0**	**1**	**0**
W2	**1**	**1**	**0**	**0**	**0**	**1**	0	**1**	**0**	**0**	**1**	**0**
END	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0

b. (1p) Milyen értékű lesz R2 az INI3 állapotban? ……**0**……..

c. (7p) A számító modullal két 4 bites szám szorzatát kell kiszámítani, **ResultL = A*B**.

A megvalósítást: ***Kezdéskor a szorzandót R1, a szorzót, R2 regiszterbe tesszük, az eredmény regisztert 0-ázzuk. A szorzást ciklusba szervezzük. Ciklusszámlálóként a speciálisan számlálóként is használható R3 regisztert használjuk (kezdőértéke C). A szorzást LSB-vel kezdjük. A szorzó LSB-jét megjelenítjük c-ben. Ha c=1, akkor az eredményként használt regiszterhez hozzáadjuk a szorzandó aktuális 2 hatványát, ha c=0, nem adjuk hozzá. Mindkét eset után előállítjuk a szorzó következő 2 hatványát és ezzel egyidőben csökkentjük a ciklusszámlálót. Ezután visszamegyünk a ciklus elejére, ahol ellenőrizzük a ciklus számlálót. Rendelje a számítás elvégzéséhez szükséges műveleteket az (egy kimenet kivételével) Moore jellegű HLSM állapotaihoz.*** Az alább felsorolt műveletekből válasszon (szükségtelenek is vannak közöttük). ***Írja az elvégzendő művelet(ek) sorszámát a megfelelő állapot neve mellé!*** Nem biztos, hogy van elvégzendő művelet, ilyenkor írjon x-et! (A HLSM gráf segítő információkat tartalmaz. A gráf pontozott részeit nem kell kitölteni, de az is segítő információ.) ***A többlet és hiányzó művelet sorszámokért pontlevonás jár!*** Egy állapothoz beírtuk, folytassa a többivel!



1: R0 = {R0[6:0],0} **2:** R0 = R0^R0 **3:** R0 ={0, R0[7:1]}
4: R0 = R0+R1 **5:** R1 = A **6:** R1 = B **7:** R1 = C
8: R1 = R0+R1 **9:** R1 = {R1[6:0],0} R2 = A **10:** R2 = B
11: R2 = C **12:** R2 = {0, R2[7:1]} **13:** R3 = R3+1
14: R3 = R3-1, **15:** R3 = R3+R1 **16:** R3=C **17:** rdy = tc3

IDLE:…**x**.. INI1: ….**2**..… INI2:…**5**… INI3: …**10**….

INI4: …**16**….TST: ..17... W1:……**12**… W2: ….**4**…..

W3:…**9, 14**…..

d. (1p) ***Adja meg a kért feltéte(lek)et a blokkvázlaton szereplő feltétel jel(ek) felhasználásával.*** (A parancs és feltétel bitek negáltját is fel lehet használni.)

f1: ……**c**…….

e. (1p) Adja meg az adatstruktúra C bemenetének (ciklus számláló kezdőértéke) értékét! (Az algoritmusban fel kell használni.)
C = ……**4**……

f. (5p) Adja meg az adatstruktúra REG_ARR regiszter tömbjének Verilog leírását a bevezetőben megadott információk alapján! Az R3 speciális számláló funkciójának megvalósításától most eltekintünk. A leírást elkezdtük, egészítse ki a hiányzó részekkel! (Az **arr** 4 db 8 bit szószélességű tároló regisztert tartalmazó tömb.)

```
module REG_ARR (input clk, wr, input [1:0] a, b,
input [7:0] in; output reg [7:0] outa, outb);

// beírás megvalósítása
reg [7:0] arr[3:0];
always@(.posedge clk)

if(wr) arr[a] <= in;
always@(...*...) // opa elő-
                 // állítása

 outa <=  arr[a];

always@(..*...) // opb elő-
                // állítása

outb <=  arr[b];
endmodule
```

IMSC1 (3p)

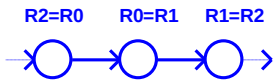
Tervezzen ***programozható modulusú 4 bites felfele számlálót***! A számláló modulusa 2-től 16-ig legyen változtatható a 4 bites d bemenetére adott szám függvényében. Addja meg az egység Verilog kódját! A számláló regiszter neve legyen q.

```
reg [3:0] q;

wire [3:0] d;

always@(posedge clk)
if(rst) q <= 4'h0;
else if(q == d) q <= 4'h0;
     else       q <= q + 4'h1;
```

IMSC2. (4p)
Készítsen az F2 feladat adatstruktúrájához egy olyan HLSM diagram részletet, amely **a regisztertömb R0 és R1 regiszterének értékét cseréli fel, a legkevesebb műveletttel**. Milyen műveletet kell beállítani az op[2:0]-on és mi legyen ez alatt dw3, up3 értéke?

Művelet: op[2:0]=111 (ALo = opb), dw3,up3 = 0,0

HLSM diagram részlet:



Maximális pontszám: 65 pont (IMSC: 7p) Rendelkezésre álló idő: 100 perc

További segítség az F2 a.-hoz

Minták különféle műveletek esetén a táblázat kitöltésére:
Ha a művelet eredménye beíródik op1-be, akkor a wr bit 1, egyébként 0. (A továbbiakban a beíródó eset szerint töltjük ki a wr bitet a táblázatban.)
Ha a művelet az R3 regiszter speciális fel vagy le számláló képességét használja: Ha R3 = R3+1, akkor dw3,up3 = 0,1. Ha R3 = R3-1, akkor dw3,up3 = 1,0. Ezek a műveletek a többi művelettel párhuzamosan végezhetők, ha a másik művelet az R3-ba nem ír. Ha (dw3,up3=0,0 vagy 1,1 az nincs hatással R3-ra.)
Ha up3,dw3 = 1,1 akkor az s[1:0]-al az A,B,C,D bemenetek közül kiválasztott külső adat lesz az ALU opb operandusa. Ekkor a regiszter tömb outa kimetére választott regiszter és egy külső adat között végzi az ALU a műveletet. (Egyébként a regiszter tömb outa és outb kimenetére választott regiszterek között.

Ri = Ri *művelet* Rj, ha a 2 operandusos műveletben a c flag nem szerepel (pl. AND, OR, EXOR).

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	j[1]	j[0]	i[1]	i[0]	k	ó	d
1	0	0	0	0	1	0	0	1	1	0	0

(*k ó d* a művelet 3 bites kódja)
Pl. R1 = R1 & R2

Ha a művelet az adatmozgatás (Ri = Rj).

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	j[1]	j[0]	i[1]	i[0]	1	1	1
1	0	0	0	0	1	0	0	1	1	1	1

Pl. R1 = R2

Ha az 1 opreandusos műveletben a c flag szerepelhet ({opa[6:0], ci&eci} és {ci&eci, opa[7:1] }) de nem akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 0-ba kell állítani és b[0] értéke közömbös (x, de a feladatokban x helyett 0 beírását kérjük).

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	x	i[1]	i[0]	k	ó	d
1	0	0	0	0	0	0	0	1	0	1	0

Pl. R1 = ({R1[6:0], 0}

Ha az 1 opreandusos műveletben a c flag szerepelhet ({opa[6:0], ci&eci} és {ci&eci, opa[7:1] }) és azt akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 1-be kell állítani és b[0] értéke közömbös (x, de a feladatokban x helyett 0 beírását kérjük).

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	x	i[1]	i[0]	k	ó	d
1	0	0	0	0	1	0	0	1	0	1	0

Pl. R1 = ({R1[6:0], c}

Ha a 2 opreandusos műveletben a c flag szerepelhet ({opa+opb+ci&eci} és {opa-opb-ci&eci}) de nem akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 0-ba kell állítani, ezért ilyenkor op2-ként csak R0 vagy R1 használható!

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	x	i[1]	i[0]	k	ó	d
1	0	0	0	0	0	0	0	1	0	0	0

Pl. R1 = R1+R0

Ha a 2 opreandusos műveletben a c flag szerepelhet ({opa+opb+ci&eci} és {opa-opb-ci&eci}) de azt akarjuk, hogy a c-t figyelembe vegye, akkor b[1]-et 1-be kell állítani, ezért ilyenkor op2-ként csak R2 vagy R3 használható!

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	x	i[1]	i[0]	k	ó	d
1	0	0	0	0	1	0	0	1	0	0	0

Pl. R1 = R1+R2+c

Ha a művelet valamelyik külső adat betöltése egy regiszterbe (Ri = K), akkor dw3,up3 == 1,1 és az s[1:0]-al a külső adatot kell kiválasztani (A esetén s[1:0]=00, B esetén s[1:0]=01, C esetén s[1:0]=10, D esetén s[1:0]=11) Ezen kívül az ALU-n az ALo=opb műveletet kell beállítani (op[2:0]=111), a wr bitet 1-be kell állítani, az a[1:0]-val kell kiválasztani, hogy melyik regiszterbe íródjon. (Az opb közömbös.)
Ha a művelet egy konstanssal való művelet, akkor az csak a műveleti kódban különbözik a fentiektől. (Példa ugyanitt az utolsósorban.)

wr	dw3	up3	s[1:0] (K)		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	k[1]	k[0]	x	x	i[1]	i[0]	1	1	1
1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0	1	1	0	0

Pl. R1 = C
Pl. R1= R1 & C

Ha z vagy c falg-et be akarjuk állítani egy művelet eredményétől függően, de az eredményre nincs szükségünk, akkor az ALU művelet beálítása mellett wr = 0-t kell beállítani. (pl. R1&B). Ekkor az eredmény nem íródik be az eredmény regiszterbe.

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	j[1]	j[0]	i[1]	i[0]	k	ó	d
0	1	1	0	1	0	0	0	1	1	0	0

(*k ó d* a művelet 3 bites kódja)
Pl. R1 & B

A végeredmény kijelzése, ha ResultL = Ri, ResultH = Rj

wr	dw3	up3	s[1:0]		b[1:0] (Rj)		a[1:0] (Ri)		op[2:0]		
11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	j[1]	j[0]	i[1]	i[0]	x	x	x
0	0	0	0	0	1	0	0	1	x	x	x

Pl. ResultL =R1, ResultH = R2