Digitális technika II. (vimia111)
5. gyakorlat: Tervezés adatstruktúra-vezérlés szétválasztással, vezérlőegység generációk

Elméleti anyag:

- Processzoros vezérlés általános tulajdonságai
  - Az induló készletben a vezérlőn kívül általános célú adatstruktúra elemek (aritmetikai-logikai egység, regiszterek stb.) is vannak
  - A külvilággal való kommunikáció BUSZ-on keresztűl történik
  - A BUSZ-on alapvetően egy rendszervezérlő, több de egyidejűleg csak egy aktív MASTER és több SLAVE van.
  - Tipikus SLAVE-ek: memória, periféria, ezek illesztése később
- Programozható processzor
  - A processzor jellemzői, HW paraméterek, SW jellemzők
  - Az utasítás architektúra és az adatstruktúra kapcsolata
  - Stack alapú, akkumulátor alapú és regisztertömbös megoldások, a felépítés hatása az utasítás kódolásra és formátumra
  - Egy egyszerű feladat Fibonacci sorozat generálása gépi szintű programjának elkészítése, az assembler és a gépi kód kapcsolata

Irodalom:

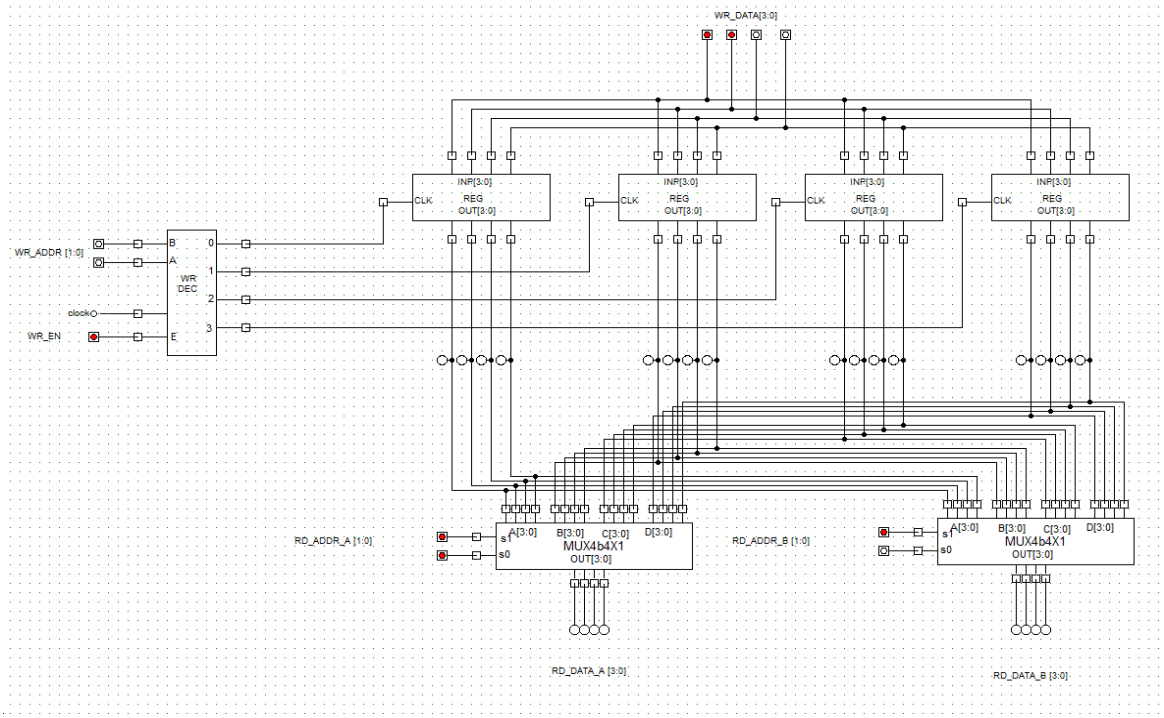
Benesóczky Zoltán: Digitális tervezés funkcionális elemekkel és mikroprocesszorokkal, egyetemi tankönyv, MK55033,.103-112.

Gyakorló példák:

5.1. 3 címes regisztertömb belső felépítése

A RISC CPU-k adatstruktúrájának jellemző eleme a 3 regiszter címes címzést (1 írási cím, 2 olvasás cím) biztosító regisztertömb, 8-16-32 bit adatszélességgel és 8-16-32 mélységű méretben. Az alkatelem használatával a két operandusú műveleteket közvetlenül, felesleges adatmozgatások nélkül lehet elvégezni, Rdest = Rsrc1 FUN Rsrc2, utasítás formátum: FUN_ DEST_ SRC1_ SRC2. Tervezzünk meg egy 4 bit széles, 4 regiszterből álló 3 cím elérésű egységet. Az interfész jelek: CLK, WR_ADR[1:0], RD_ADR_A[1:0], RD_ADR_B[1:0], WR_EN, WR_DATA[3:0], RD_DATA_A[3:0], RD_DATA_B[3:0].

A terv 4 db élvezérelt DFF-ból álló 4 bites regisztert tartalmaz. A RST jelet nem használjuk, mivel a processzorok regisztereire a RST alapvetően nincs hatással, tehát a programozó nem tételezheti fel, hogy a regiszterek induláskor bármilyen alapértékkel rendelkeznek. Mivel a DW DFF-ok-nak nincs CE jelük, ezért a bemeneti MUX-os adatválasztás helyett az órajelet kapuztuk a WR_DEC makróban. Ez nem szép megoldás, de most fogadjuk el. Egy szebb terveben, inkább szinkron töltéssel rendelkező regisztereket rajzolnánk, és a dekóder ezeket vezérli, míg az órajel közös és globális, azaz nem kapuzott.
A REG_FILE 3 regiszter címet képes egyszerre kezelni, tehát támogatja az olyan utasításkészletet, amelyben 3 operandus szerepel, vagyis Rdest = Rsrc1 OPER Rscr2. Ez hasznos, mert nagyon ritkán kell csak regiszter másolásokat végeznünk. Ugyanakkor 32 méretű regiszter tömb esetén ehhez 3x5 bit, azaz 15 utasításbit kell, ami csak 32 bites utasításformátum mellett engedhető meg.



Megjegyzés1: A fenti megoldásban ugyanazt a címet használva mind a WR_ADR, mind a RD_ADR_A bemeneten is, a szintén elterjedt 2 címes regisztertömböt kapjuk, ahol a művelet eredménye mindig felülírja az egyik operandust : Rsrc1 = Rsrc1 FUN Rsrc2. Itt a programba gyakran kénytelenek vagyunk extra adatmozgatásokat beépíteni (ha pl. szeretnénk megőrizni a Rsrc1 eredeti értékét). Ugyanígy egy ilyen megoldásnál gondolnunk kell a SUB kivonás (A-B) ALU művelet mellett a fordított kivonás RSUB (B-A) ALU művelet megvalósítására is, vagy a (B-A) kivonáshoz 2 utasítás kell (1 regisztermásolás és egy kivonás).

Megjegyzés2: A fenti megoldásban a WR_ADR címet használva a RD_ARD_A bemeneten is és a RD_ADR_B címet elhagyva a szintén elterjedt 1 címes regisztertömböt kapjuk, ami az un. akkumulátor típusú adatstruktúrák jellemző elem. Ebben az esetben a művelet a következő lehet: ACC = ACC FUN Rsrc. Ez olcsó, de rengeteg extra adatmozgatást kíván a regiszterek és az akkumulátor között. A regiszterek írása gyakran csak az Rdst = ACC áttöltéskor lehetséges, esetleg még az Rdst = Rdst +1, vagy Rdst-1 esetekben.

Megjegyzés3: Hangsúlyozzuk, hogy az olvasás a cím megjelenésére azonnali, közvetlen, aszinkron végrehajtódik (kombinációs jelutak), míg az írás parancs az adott címen az órajelciklus végén, a felfutó élre (szinkron) hajtódik végre. Emiatt tehát lehetséges egyetlen utasításon belül a regiszter értékének használata és az új érték visszaírása (Read-Modify-Write)

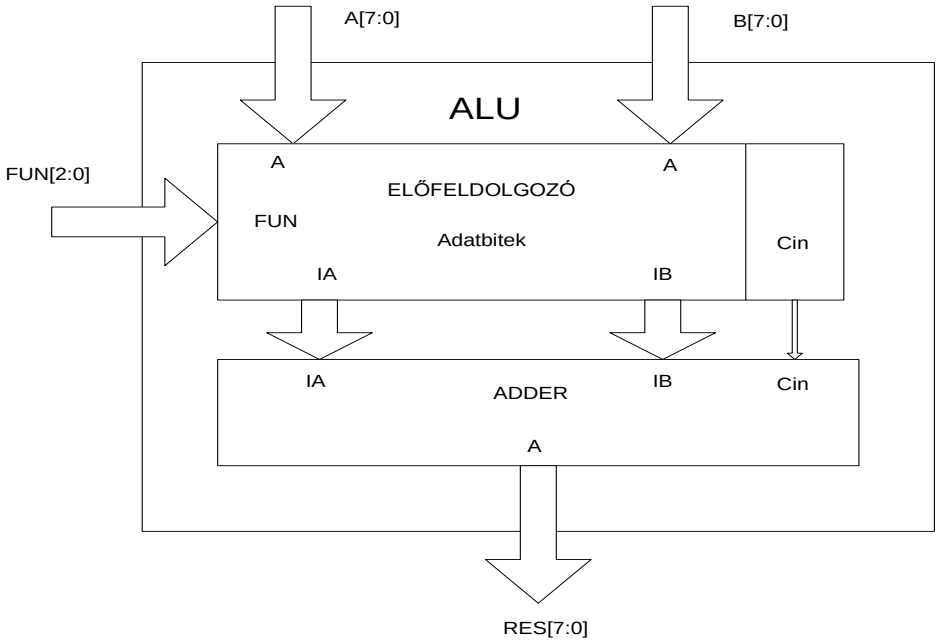
A tervezői fájlok: REG_FILE4x4.dwm, FILE_REG.dwm, MUX4b4x4.dwm, WR_DEC2x4.dwm, mellékelve.

5.2. ALU egység tervezése

Az adatstruktúra másik fontos eleme az ALU. A szükséges művelethalmaz (aritmetikai, logikai, shift) kiválasztása gondos analízist igényel, a hatékonyságot egyértelműen meghatározza. Tervezzünk egy ALU egységet, ami a következő műveleteket biztosítja:

FUN[2:0]	Funkció	Leírás
000	A+B	Összeadás
001	A-B	Kivonás
010	A+1	Inkrementálás
011	A	Adatátmásolás
100	A and B	Bitenkénti AND
101	A or B	Bitenkénti OR
110	A xor B	Bitenkénti XOR
111	not A	Bitenkénti NOT

A triviális megoldás a 8 funkció önálló realizálása és egy nagyméretű 8 bites adatokon dolgozó 8 bemenetű multiplexer használata. Rajzoljuk fel vázlatosan, de magyarázzuk el, hogy ez nagyon gazdaságtalan.
Egy jobb megoldást eredményez az alábbi összeadóra és elő-feldolgozóra alapuló struktúra. Ennek megfelelően a kimeneti eredményt mindig az összeadó kimenete adja, és a bemeneti operandusokat A[7:0], B[7:0] egy átalakítás után vezetjük az összeadó IA[7:0], IB[7:0] bemeneteire.



A fenti blokkvázlat alapján az előfeldolgozó által szükséges műveletek a belső (IA[7:0], IB[7:0]) adatkimenetek és a Cin jel generálását jelentik. Amennyiben az eredetileg tervezett notA művelet helyett a notB műveletet valósítjuk meg, a hálózat egyszerűbb lehet (mivel ez a részfeladat része az A-B műveletnek is). Így a műveleti tábla az elő-feldolgozó funkcióit is részletezve a következő:

FUN[2:0]	Funkció	Leírás	IA	IB	Cin
000	A+B	Összeadás	A	B	0
001	A-B	Kivonás	A	not B	1
010	A+1	Inkrementálás	A	0	1

011	A	Adatátmásolás	A	0	0
100	A and B	Bitenkénti AND	0	A and B	0
101	A or B	Bitenkénti OR	0	A or B	0
110	A xor B	Bitenkénti XOR	0	A xor B	0
111	not A not B	Bitenkénti NOT	0	not A notB	0

Egyszerűen kiolvasva:
IA = A, ha FUN[2] = =0, egyébként 0, tehát ez egy 2 bemenetű ÉS kapu minden bitre.
Cin = 1 ha FUN == 1 vagy FUN= 2, tehát ez kettő db három bemenetű ÉS kapu .
IB = Ez már bonyolultabb, de azért megtervezhető. 5 bemenet, 1 kimenet minden bitre.

FUN[2:0]	IBx, x=3…0
000	Bx
001	not Bx
010	0
011	0
100	Ax and Bx
101	Ax or Bx
110	Ax xor Bx
111	not B

A feladathoz nem készült DW rajz, felesleges, mindenki el tudja képzelni.

Megjegyzés: A műveletek kiválasztása véletlenszerűen, egy példa alapján történt. Megemlíthető, hogy nem feltétlenül optimális, mivel ha az utasításban tudunk közvetlenül adatot megadni (és ez általában teljesül, mint konstans operandus), akkor az A+1 vagy a notA (notB) felesleges. Az inkrementáláshoz használható a B bemeneten az 1, mint konstans, a negálás pedig az A XOR FF, azaz csupa egyes konstans. Tehát egy ALU funkciókészlet kialakítása gondos tervezést igényel.

5.3 Feltételjelek

A CPU vezérlőegysége a program következő utasításának címét feltételes utasítás esetén pl. az aktuális ALU művelet eredménye alapján határozza meg. Az aritmetikai típusú ALU műveletekben mindig **kettes komplemens** számábrázolást tételezünk fel, mind a bemeneten, mind a kimeneten.
A tipikus feltételjelek a következők:

Z	Az ALU művelet eredménye nulla
C	Az ALU művelet végrehajtásakor átvitel keletkezett
N	Az ALU művelet eredményének előjele
V	Az ALU művelet során túlcsordulás keletkezett

Tervezzük meg a feltétel jelek logikai hálózatát!

1. Z = RES[7:0]==0, azaz a hálózat egy 8 bemenetű NOR kapu.
2. C= Cout4, az ALU ADDER átvitelbit közvetlenül felhasználható
3. N= RES[7], az eredmény legnagyobb helyiértékű bitje közvetlenül felhasználható.
4. V= ? Mikor van túlcsordulás kettes komplemensű számokkal végzett műveleteknél?

Túlcsordulás csak aritmetikai műveletek esetén fordulhat elő.
Azonos előjelű operandusok esetén:
Összeadásnál is és kivonásnál is, ha az operandusok előjele azonos, de az eredmény előjele ettől eltérő, akkor túlcsordulás van (POS + POS = NEG, vagy NEG + NEG = POS).

Eltérő előjelű operandusoknál a helyzet nem ilyen egyértelmű.
Összeadásnál soha nem keletkezik túlcsordulás (POS + NEG vagy NEG + POS eredménye mindig helyes), ekkor soha nem lépünk ki a tartományból.

Kivonásnál a NEG –POS vagy a POS–NEG művelet eredménye előjel szerint tetszőleges lehet, de sajnos túlcsordulás ezzel együtt is előfordulhat. (Pl. -100 – (+5) = -105, -100 – (+50) = -150, de ez kisebb lenne mint -128, tehát = +106). Vagyis nem elegendő az előjel eltérése, az operandusok pozíciója és értéke is fontos. Ha az előjel azonos az első operandus előjelével, akkor a művelet eredménye helyes. Ez azonban a kimeneti feltétel jelet a bemeneti operandusok előjeléből számítja.

Összefoglalóan a kettes komplemens ADD/SUB áramkörben az OVR bitet a legegyszerűbben a Cout7 XOR Cout6 hálózattal állíthatjuk elő, vagyis az utolsó előtti és az utolsó bitpozíciók átvitelbitjeit kell használnunk.

5.4 A DW alkatrészkönyvtár RTL (Regiszter Transzfer Szintű) Verilog megvalósítása

A HDL nyelvű leírásokban általában nem strukturális szintű specifikáció a legmegfelelőbb. A megismert DW könyvtár elemeinek elkészítettük az RTL szintű Verilog HDL specifikációit is. Ezeket, és a korábbi segédlet tartalmát egyetlen táblázatba foglaltuk, a következő módon:

DigitalWorks Funkcionális alkatrész katalógus v1.0
Verilog HDL példa megvalósítása közvetlen strukturális, kapcsolási rajz szintű és RTL szintű leírással
2012

A strukturális leírásokban az egyes alkatrészek specifikációjánál törekedtünk a DW könyvtár megfelelő kapcsolási rajzainak és építkezési szabályainak lehető legjobb másolására, akkor is, ha ez a HDL specifikáció szempontjából nem a legkedvezőbb stílus használatához vezetett
Az RTL szintű leírásokban a szokásos legkedvezőbb leírási stílust használtuk

STRUKTURÁLIS LEÍRÁS

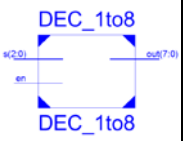
```
////////////////////////////////////////////////////
// DW könyvtár 1.0 DEC_1to8
// 1-ből 8-as Demultiplexer vagy 3-ból 8-as Dekóder
////////////////////////////////////////////////////
module DEC_1to8(
    input en,
    input [2:0] s,
    output [7:0] out
    );

////////////////////////////////////////////////////
// Kapu szintű leírás, 8 db 4 bemenetű ÉS logika,
// ponált/negált bemenetekkel
////////////////////////////////////////////////////

assign out[0] = en & ~s[2] & ~s[1] & ~s[0];
assign out[1] = en & ~s[2] & ~s[1] &  s[0];
assign out[2] = en & ~s[2] &  s[1] & ~s[0];
assign out[3] = en & ~s[2] &  s[1] &  s[0];
assign out[4] = en &  s[2] & ~s[1] & ~s[0];
assign out[5] = en &  s[2] & ~s[1] &  s[0];
assign out[6] = en &  s[2] &  s[1] & ~s[0];
assign out[7] = en &  s[2] &  s[1] &  s[0];

endmodule
```

SZIMBÓLUM



RTL SZINTŰ LEÍRÁS

```
////////////////////////////////////////////////////
// DW könyvtár RTL DEC_1to8
// 1-ből 8-as Demultiplexer vagy 3-ból 8-as Dekóder
////////////////////////////////////////////////////
module DEC_1to8(
    input en,
    input [2:0] s,
    output [7:0] out
    );

////////////////////////////////////////////////////
// Egy dekóder logika shifteléssel
////////////////////////////////////////////////////

assign out = {7'b0,en} << s;

endmodule
```

A teljes dokumentum a tárgy honlapon megtalálható. A vizsgán az RTL szintű leírás megadása vagy felismerése kérdésként szerepelhet.