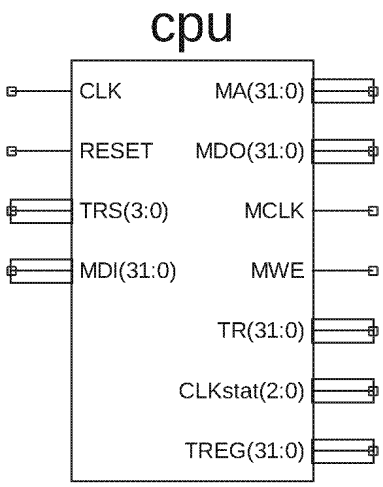
Mikroprocesszor

A mikroprocesszor egy RISC felépítésű (LOAD/STORE), Neumann architektúrájú 32 bites soft processzor, amelyet FPGA-val valósítunk meg.



A mikroprocesszor részei

A mikroprocesszor a szokásos építőelemekből áll.

Ütemező, vezérlő

Az ütemező állítja elő a vezérlő jeleket a processzor többi eleme számára. Az ütemező állapot kódokat állít elő a működési fázisok jelzésére, az állapotok a **`CLK`** bemenetre adott órajel hatására követik egymást. Az állapotok bináris sorszáma a **`CLKstat`** kimeneten jelenik meg. A **`RESET`** bemenet az ütemezőt alaphelyzetbe állítja.
A vezérlőhöz tartozik még az ún. utasítás regiszter, amely a beolvasott utasítást tárolja a végrehajtás idejére.

Memória illesztő

Mivel a processzor Neumann architektúrájú, ezért csak egy memória illesztővel rendelkezik. Ezen az illesztőn keresztül olvassa be az utasításokat, és ezen az illesztőn keresztül végzi el memória írási és olvasási műveleteket is.
Az illesztő az **`MA`** kimeneteken adja ki a művelethez használandó memória címet (32 bites). Írás esetén a memóriába tárolandó adat az **`MDO`** kimeneteken jelenik meg (32 bit). Ebben az esetben az **`MWE`** kimenet magas szintű lesz (írás jelzés).
Olvasás műveletnél a megcímzett memória tartalmát az **`MDI`** bemenetre kell juttatni, ekkor az **`MWE`** kimenet alacsony lesz. Mind írás, mind olvasás esetén megjelenik az **`MCLK`** kimeneten egy impulzus, ennek felfutó éle használható a memória számára beíró jelként, illetve a pufferelt kimenetű memóriák esetén a kimeneti puffer beírására.

Memória szervezés

A processzor által kezelt memória 32 bites szélességű, minden 32 bites rekesznek külön címe van. Memória műveletnél a processzor a megcímzett rekeszből 32 bites adatot olvas be, illetve 32 bites adatot ír ki.

Regiszterek

A processzor állapota a belső regiszterekben tárolódik. A regiszter készlet 16 db 32 bites regiszterből áll. A regiszterek elnevezése: `R0`, `R1`...`R15`.

Speciális regiszterek

PC

A processzor az `R15` regisztert használja program számlálóként (Program Counter), ezért ennek tartalma minden utasítás végrehajtásakor eggyel növekszik.

LR

A `CALL` utasítás a visszatéréshez szükséges címet az `R14` (Link Regiszter) regiszterbe menti.

SP

A processzor nem használ verem műveleteket, a verem megszervezése a programozó feladata. Az ehhez szükséges verem mutató (Stack Pointer) céljára bármelyik regiszter felhasználható. A programok megírásakor az `R13` regisztert használtuk SP-ként.

Aritmetikai-logikai egység

Az adat manipulációs műveleteket az ALU végzi, amely 32 bites egész (előjeles és előjel nélküli) adatokkal tud műveleteket végezni. Az ALU-nak 3 adat bemenete van: Rd, Ra, Rb. Az Rd bemenetre az eredményt eltároló regiszter eredeti (művelet előtti) értékét kell kötni. Az Ra és Rb bemenetekre a művelet operandusaként használt regiszterek értéke kerül. Az egy operandusú műveletek csak az Ra bemenetet használják, míg a két operandusú műveletek az Ra és az Rb értékeket használják fel. Az operandus nélküli műveleteknél az eredmény a változatlan Rd bemeneti érték lesz.

FLAG regiszter

Az aritmetikai egység a műveletek eredményének jellemzőit a FLAG regiszterben tárolja el. Ez a regiszter nem része az általános regiszter készletnek, a tartalmához az utasítások speciális módon férhetnek csak hozzá.

Carry flag (C)

Aritmetikai műveletek után azt jelzi, hogy előjel nélkülinek tekintve az operandusokat, volt-e túlcsordulás (1: volt, 0: nem volt).

Overflow flag (O)

Aritmetikai műveletek után azt jelzi, hogy előjelesnek tekintve az operandusokat, volt-e túlcsordulás (1: volt, 0: nem volt).

Sign flag (S)

Aritmetikai műveletek után után az eredmény előjelét jelzi (1: negatív, 0: pozitív)

Zero flag (Z)

Aritmetikai és logikai műveletek után azt jelzi, hogy az eredmény nulla-e (1: igen, 0: nem)

A mikroprocesszor működése

Az utasítások végrehajtása a következő fázisokból áll.

Fetch

Ebben a fázisban a processzor elvégzi a soron következő utasítás beolvasását. Az **MA** kimeneten megjelenik az `R15` regiszter tartalma, az **MWE** kimenet alacsony szintű lesz, míg az **MCLK** kimeneten megjelenik egy felfutó él.

Dekódolás

A következő fázisban fetch során megcímzett memória tartalom megjelenik az **MDI** bemeneten, ezt a processzor eltárolja az utasítás regiszterben. Ezzel egyidejűleg az `R15` tartalma eggyel nő.

Számítás

Az utasítás regiszter által meghatározott regiszterek tartalma megjelenik az ALU bemenetein, ezek az adatok áthaladnak az ALU-ban lévő aritmetikai hálózatokon. Ezek mindegyike kombinációs (nem sorrendi) hálózat, de több szintű felépítésük miatt a kimeneti eredmények megjelenéséhez több idő kell. Ennek kivárására szolgál a “számítás” fázis. Az utasítás regiszter tartalma azt is meghatározza, hogy az ALU kimenetén melyik hálózat eredménye jelenjen meg.

Memória művelet

Ha az utasítás memória műveletet ír elő, akkor az ebben a fázisban zajlik le. Az **MA** kimeneten az utasításban meghatározott regiszter tartalma jelenik meg címként, írás esetén

az **MDO** kimenetekre az utasításban megadott (kiírandó) regiszter tartalma kapcsolódik. Az **MWE** kimenet az elvégzendő műveletnek megfelelő értékű lesz. Az **MCLK** kimeneten megjelenik egy felfutó él. Olvasáskor a memóriának a címzett rekesz tartalmát az **MDI** bemenetre kell kapcsolnia, és ott kell tartania a következő fázis ideje alatt is. Ezért az **MA** kimeneteken a cím fennmarad a következő fetch fázis elejéig.
Ha nincs szükség memória műveletre, akkor ebben a fázisban semmi sem történik. Ez megnöveli az aritmetikai eredmények kiszámításához rendelkezésre álló időt.

Eredmény tárolás

Az utasítás eredménye vagy az ALU kimenetén megjelent adat, vagy memória olvasásnál az **MDI** bemenetek állapota. Az eredmény tárolási (visszaírási, Write-Back) fázisban ez az érték beíródik az utasítás által meghatározott regiszterbe. Néhány utasításnak nincs eredménye, ezeknél az írás nem zajlik le (pl. memóriába írás, “nincs művelet” utasítás). Aritmetikai utasításoknál az ALU által előállított jelzőbitek is beíródnak a flag regiszterbe.

Utasítások

A processzor utasítás készlete 8 utasításból áll. Minden utasítás kódja 32 bites, egy memória rekeszben tárolódik.

Feltételes végrehajtás

Minden utasítás ellátható egy feltétellel, ekkor a memória írási és a visszaírási műveletek csak akkor hajtódnak végre, ha a feltétel teljesül. A feltétel megadja, hogy mely jelzőbit milyen értéke esetén lesz igaz. A feltétel elhelyezkedése az utasítás kódjában a következő:

```
FFVC.---- ----.---- ----.---- ----.----
```

Az `F` jelű bitek jelölik ki a flag bitet:

- `00`: S jelzőbit
- `01`: C jelzőbit
- `10`: Z jelzőbit
- `11`: O jelzőbit

A feltétel akkor teljesül, ha a kiválasztott jelzőbit értéke egyezik az utasításban lévő `V` bit értékével. A feltételt a vezérlő csak akkor veszi figyelembe, ha a kódban a `C` jelű bit `1`, egyébként az utasítás feltétel nélkül hajtódik végre.

Utasítások

NOP

“Nincs művelet” utasítás, hatására a memória, és a visszaírási fázisban nincs művelet.

- Művelet: nincs

- Kódja: `FFVC.0000 DDDD.AAAA BBBB.OOOO.OXXX XXXX.XXXX`
  Az `FFVC` bitek a végrehajtás feltételét adják meg, a `DDDD`, `AAAA` és `BBBB` bitek 4 bites bináris kóddal megadják az ALU 3 bemeneteként felhasznált regiszterek sorszámát. Az `OOOOO` bitek az ALU műveletei közül választanak ki egyet.
- Módosított jelzőbitek: -

Az utasítás diagnosztikai célra használható, segítségével meg lehet figyelni az ALU be- és kimeneti értékeit.

LD Rd,Ra

LOAD memory to register. Memória tartalom regiszterbe való beolvasására használható utasítás.

- Művelet: `Rd:= mem[Ra]`
- Kódja: `FFVC.0001 DDDD.AAAA XXXX.XXXX XXXX.XXXX`
  A `DDDD` bitek az Rd regiszter, míg az `AAAA` bitek az Ra regiszter sorszámát adják meg.
- Módosított jelzőbitek: -

ST Rd,Ra

STORE register to memory. Egy regiszter memóriába írására használható utasítás.

- Művelet: `mem[Rd]:= Ra`
- Kódja: `FFVC.0010 DDDD.AAAA XXXX.XXXX XXXX.XXXX`
- Módosított jelzőbitek: -

MOV Rd,Ra

MOVE register to register. Regiszter másolás utasítás, az Ra regiszter értékét átmásolja az Rd regiszterbe.

- Művelet: `Rd:= Ra`
- Kódja: `FFVC.0011 DDDD.AAAA XXXX.XXXX XXXX.XXXX`
- Módosított jelzőbitek: -

LDL0 Rd,adat

LOAD low half, set high half 0. Konstans adat regiszterbe írásához használható, az adat az utasításban található. Az adat 16 bites, a megadott regiszter alsó helyi értékű felére kerül, a regiszter felső helyi értékű fele 0 lesz.

- Művelet: `Rd:= 0.adat`

- Kódja: `FFVC.0100 DDDD.XXXX LLLL.LLLL LLLL.LLLL`
  A `DDDD` bitek választják ki a módosítandó regisztert, az `L` jelű bitek tartalmazzák a 16 bites konstanst.
- Módosított jelzőbitek: -

LDL Rd,adat

LOAD low half of register. Konstans adat regiszterbe írásához használható, az adat az utasításban található. Az adat 16 bites, a megadott regiszter alsó helyiértékű felére kerül, a regiszter felső helyiértékű fele nem változik.

- Művelet: `Rd:= High(Rd).adat`
- Kódja: `FFVC.0101 DDDD.XXXX LLLL.LLLL LLLL.LLLL`
  A `DDDD` bitek választják ki a módosítandó regisztert, az `L` jelű bitek tartalmazzák a 16 bites konstanst.
- Módosított jelzőbitek: -

LDH Rd,adat

LOAD high half of register. Konstans adat regiszterbe írásához használható, az adat az utasításban található. Az adat 16 bites, a megadott regiszter felső helyi értékű felére kerül, a regiszter alsó helyi értékű fele nem változik.

- Művelet: `Rd:= adat.Low(Rd)`
- Kódja: `FFVC.0110 DDDD.XXXX LLLL.LLLL LLLL.LLLL`
  A `DDDD` bitek választják ki a módosítandó regisztert, az `L` jelű bitek tartalmazzák a 16 bites konstanst.
- Módosított jelzőbitek: -

CALL cím

Szubrutin hívás. Mivel a processzor memóriában lévő vermet nem kezel, a visszatérési címet (az R15 megnövelt értékét, amely a CALL-t követő utasítás címét tartalmazza) a memória fázisban átmásolja, beírja az R14 regiszterbe. Egymásba ágyazott szubrutin hívások esetén az R14 mentéséről a programozónak kell gondoskodnia.

- Művelet: `R14:= R15; R15:= cím`
- Kódja: `FFVC.1AAA AAAA.AAAA AAAA.AAAA AAAA.AAAA`
  Az utasításban szereplő A bitek az `R15` alsó helyi értékű 27 bitjébe kerünekl. A legfelső 5 helyi érték `0` lesz.
- Módosított jelzőbitek: -

Az utasítás kódjában csak 27 bites cím helyezhető el, ezért csak a memória tartomány első 32-ed részében lévő szubrutint lehet meghívni.

A szubrutinból való visszatéréshez az R14 (LR) értét kell bemásolni az R15-be (PC) egy MOV utasítással.

Aritmetikai, logikai utasítások

Ezek az utasítások az ALU kimenetén megjelenő adatot használják a visszaírási fázisban. Az utasítások kódja:

```
FFVC.0111 DDDD.AAAA BBBB.OOOO OXXX.XXXX
```

Az `FFVC` bitek a végrehajtási feltételt adják meg, a `DDDD` az eredményt tároló regiszter száma, az `AAAA` és `BBBB` bitek pedig az operandusokat tartalmazó regiszterek sorszámai. Az `OOOOO` bitek választják ki, hogy az ALU hálózatai közül melyik eredményét használjuk. Ez alapján maximum 32 művelet lehetséges, de ennél jelenleg kevesebb van megvalósítva.

ADD Rd,Ra,Rb

Összeadás átvitel nélkül.

- Művelet: `Rd:= Ra+Rb`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0000 0XXX.XXXX`
- Módosított jelzőbitek: C,Z,S,O

ADC Rd,Ra,Rb

Összeadás átvitellel.

- Művelet: `Rd:= Ra+Rb+C`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0000 1XXX.XXXX`
- Módosított jelzőbitek: C,Z,S,O

SUB Rd,Ra,Rb

Kivonás átvitel nélkül.

- Művelet: `Rd:= Ra-Rb`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0001 0XXX.XXXX`
- Módosított jelzőbitek: C,Z,S,O

SBB Rd,Ra,Rb

Kivonás átvitellel.

- Művelet: `Rd:= Ra-Rb-C`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0001 1XXX.XXXX`

- Módosított jelzőbitek: C,Z,S,O

INC Rd,Ra

Növelés eggyel. Az Ra regiszter eggyel növelt értéke az Rd regiszterbe kerül. Az Ra és az Rd ugyanaz is lehet.

- Művelet: `Rd:= Ra+1`
- Kódja: `FFVC.0111 DDDD.AAAA XXXX.0010 0XXX.XXXX`
- Módosított jelzőbitek: C,Z,S,O

DEC Rd,Ra

Csökkentés eggyel. Az Ra regiszter eggyel csökkentett értéke az Rd regiszterbe kerül. Az Ra és az Rd ugyanaz is lehet.

- Művelet: `Rd:= Ra-1`
- Kódja: `FFVC.0111 DDDD.AAAA XXXX.0010 1XXX.XXXX`
- Módosított jelzőbitek: C,Z,S,O

AND Rd,Ra,Rb

Bitenkénti és művelet. Az eredmény egyes helyi értékei a két operandus ugyanazon helyi értékein lévő két bit közötti logikai ÉS művelet eredményeként keletkeznek. Bit(ek) vizsgálatára, illetve bit(ek) 0-ba állítására használható.

- Művelet: `Rd[i]:= Ra[i]&Rb[i]`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0011 0XXX.XXXX`
- Módosított jelzőbitek: Z

OR Rd,Ra,Rb

Bitenkénti vagy művelet. Az eredmény egyes helyi értékei a két operandus ugyanazon helyi értékein lévő két bit közötti logikai VAGY művelet eredményeként keletkeznek. Bit(ek) 1-be állítására használható.

- Művelet: `Rd[i]:= Ra[i]|Rb[i]`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0011 1XXX.XXXX`
- Módosított jelzőbitek: Z

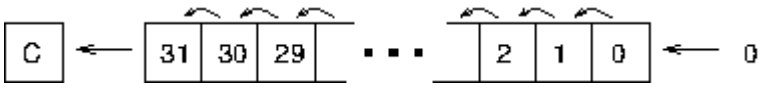
XOR Rd,Ra,Rb

Bitenkénti kizáró vagy művelet. Az eredmény egyes helyi értékei a két operandus ugyanazon helyi értékein lévő két bit közötti logikai KIZÁRÓ VAGY művelet eredményeként keletkeznek. Bit(ek) összehasonlítására, illetve negálására használható.

- Művelet: `Rd[i]:= Ra[i]^Rb[i]`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0100 0XXX.XXXX`
- Módosított jelzőbitek: Z

SHL Rd,Ra

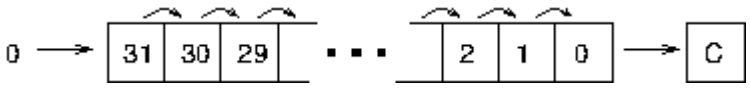
Bitenkénti (logikai) eltolás, egy bittel balra. Az Ra regiszter minden bitje egy helyi értékkel feljebb lép, a 0-ik bit 0 lesz. A legfelső helyi értékű bit a C jelzőbitbe másolódik. Az eredmény az Rd regiszterbe kerül.



- Művelet: `C:= Ra[31]; Rd[i]:= Ra[i-1]; Rd[0]= 0`
- Kódja: `FFVC.0111 DDDD.AAAA XXXX.0100 1XXX.XXXX`
- Módosított jelzőbitek: C,Z

SHR Rd,Ra

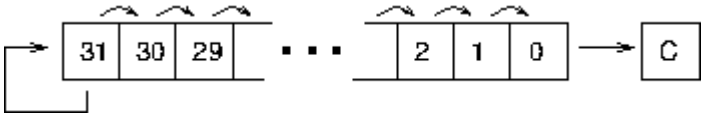
Bitenkénti (logikai) eltolás, egy bittel jobbra. Az Ra regiszter minden bitje egy helyi értékkel lejjebb lép, a 31-ik bit 0 lesz. A legalsó helyi értékű bit a C jelzőbitbe másolódik. Az eredmény az Rd regiszterbe kerül.



- Művelet: `C:= Ra[0]; Rd[i]:= Ra[i+1]; Rd[31]= 0`
- Kódja: `FFVC.0111 DDDD.AAAA XXXX.0101 0XXX.XXXX`
- Módosított jelzőbitek: C,Z

SHA Rd,Ra

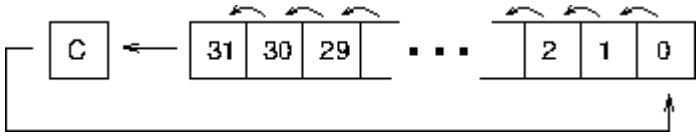
Bitenkénti (aritmetikai) eltolás, egy bittel jobbra. Az Ra regiszter minden bitje egy helyi értékkel lejjebb lép, a 31-ik bit nem változik. A legalsó helyi értékű bit a C jelzőbitbe másolódik. Az eredmény az Rd regiszterbe kerül.



- Művelet: `C:= Ra[0]; Rd[i]:= Ra[i+1]; Rd[31]= Ra[31]`
- Kódja: `FFVC.0111 DDDD.AAAA XXXX.1000 0XXX.XXXX`
- Módosított jelzőbitek: C,Z

ROL Rd,Ra

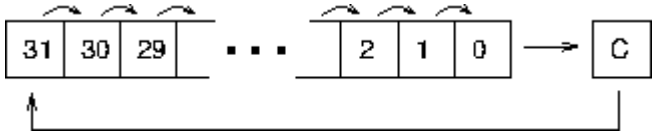
Forgatás egy bittel balra, a C jelzőbiten keresztül. Az Ra regiszter tartalmának minden bitje egy helyi értékkel balra lép. A legfelső helyi értékű bit a C jelzőbitbe kerül, aminek az eredeti értéke lép be a legalsó helyi értékre. Az eredmény az Rd regiszterbe kerül.



- Művelet: `C:= Ra[31]; Rd[i]:= Ra[i-1]; Rd[0]= C`
- Kódja: `FFVC.0111 DDDD.AAAA XXXX.0101 1XXX.XXXX`
- Módosított jelzőbitek: C,Z

ROR Rd,Ra

Forgatás egy bittel jobbra, a C jelzőbiten keresztül. Az Ra regiszter talmának minden bitje egy helyi értékkel jobbra lép. A legalsó helyi értékű bit a C jelzőbitbe kerül, aminek az eredeti értéke lép be a legfelső helyi értékre. Az eredmény az Rd regiszterbe kerül.



- Művelet: `C:= Ra[0]; Rd[i]:= Ra[i+1]; Rd[31]= C`
- Kódja: `FFVC.0111 DDDD.AAAA XXXX.0110 0XXX.XXXX`
- Módosított jelzőbitek: C,Z

MUL Rd,Ra,Rb

32 bites adatok szorzása. A 64 bites eredmény alsó 32 bitjét számolja ki.

- Művelet: `Rd:= Ra*Rb`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0110 1XXX.XXXX`
- Módosított jelzőbitek: C,Z,S,O

DIV Rd,Ra,Rb

32 bites adatok osztása, az eredmény egész részét számolja ki.

- Művelet: `Rd:= Ra/Rb`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0111 0XXX.XXXX`
- Módosított jelzőbitek: C,Z,S,O

CMP Rd,Ra,Rb

Összehasonlítás. A művelet a SUB utasítással egyezik, de az eredmény nem íródik be az Rd regiszterbe, csak a jelzőbitek tárolódnak el.

- Művelet: `Rd:= Rd; FLAGS:= Ra-Rb`
- Kódja: `FFVC.0111 DDDD.AAAA BBBB.0111 1XXX.XXXX`
  Az utasításban meg kell adni az Rd regiszter sorszámát is (`DDDD` bitek), az eredmény ennek a regiszternek a tartalma lesz, amely saját magába íródik vissza.
- Módosított jelzőbitek: C,Z,S,O

SETC

A C jelzőbit 1-be állítása.

- Művelet: `C:= 1`
- Kódja: `FFVC.0111 XXXX.XXXX XXXX.1000 1XXX.XXXX`
- Módosított jelzőbitek: C

CLRC

A C jelzőbit 0-ba állítása.

- Művelet: `C:= 0`
- Kódja: `FFVC.0111 XXXX.XXXX XXXX.1001 0XXX.XXXX`
- Módosított jelzőbitek: C

Diagnosztika

A processzor működése nyomon követhető a teszt kimeneteken megjelenő jelek megfigyelésével.

CLKstat

A `CLKstat` kimenetek az egymás utáni fázisok sorszámát adják bináris kódban.

TREG

A `TREG` kimeneteken a 16 regiszter egyikének értéke nyerhető ki, a regiszter sorszámát a `TRS` bemenetekre kell kapcsolni, 4 bites bináris kódban.

TR

A `TR` kivezetéseken a processzor valamely belső adatát lehet megjeleníteni, a megfelelő adatot a `TRS` bementekre adott 4 bites bináris kóddal lehet kiválasztani.

TRS bemenet értke	TR kimeneteken megjelenő adat
0	R15 (PC) értéke
1	R14 (LR) értéke
2	R13 regiszter értéke (általában SP)
4	IC (utasítás regiszter) értéke
5	ALU kimenete
6	Utasítás eredménye
7	Visszaírásnál használt adat
8	ENA (utasítás feltétel értéke) Rd, Ra, Rb regiszterek sorszámai
9	Az Rd regiszter értéke
10 (A)	Az Ra regiszter értéke
11 (B)	Az Rb regiszter értéke
12 (C)	Az ütemező kimenő vezérlő jelei
13 (D)	MA (memória illesztő cím kimenete)
14 (E)	MDO (memória illesztő adat kimenete)
15 (F)	MDI (memória illesztő adat bemenete)