# HexMG fájlformátum

Ha a sor szóközzel vagy \*-gal kezdődik, akkor a sor komment. ;-től kezdve a sor maradék része komment.

Az '-okat figyelmen kívül hagyja a fájlban. (A használatuk valójában csak számliterálisokban megengedett, de bárhol figyelmen kívül hagyja a program.) Egyéb helyeken hibát okozhat, ezért ne tegyük pl. // közé (mármint a per jelek közé), whitespace mellé és a sor végére sem.

## .SUNREDTREE

.SUNREDTREE fa

* a fa neve az, hogy fa, de lehet simán egy szám is

RED 1 0 0 0 0 1

* CÉLSZINT CÉLCELLAINDEX FORRÁSSZINT\_1 FORRÁSCELLAINDEX\_1 FORRÁSSZINT\_2 FORRÁSCELLAINDEX\_2
* A CÉLSZINT sorrendje tetszőleges, akár vegyesen következhetnek egymás után különböző szintű redukciók, pl. ha a fa egyik ágát szeretnénk előbb összerakni, aztán a másikat, azt is lehet. 1-től indul, a 0. szint a komponensek szintje. A végére lehetőleg ne maradjon ki index, de ha kimarad, akkor is működik.
* A CÉLCELLAINDEX az adott célszinten belül kötelezően mindig a következő szabad index (0-tól indul). Azaz a szimulátornak csak ellenőrzésre szolgál, ill. a netlistát olvasónak segít követni a kódot. Azaz nem lehet tetszőleges sorrendben definiálni a célcellákat.
* A forráscellák bármely, a célszintnél kisebb indexű szintről származhatnak, azon belül bármelyik lehet. A két forráscella sorrendje is tetszőleges. Egy cella forrásként csak egyszer szerepelhet.
* A legmagasabb indexű célszinten pontosan egy redukció lehet, és az ez alá felépülő fában minden korábban definiált cellának pontosan egyszer benne kell lennie.

.END SUNREDTREE fa

* A blokk lezárása mellett alapvetően ellenőrzésre szolgál, hogy oda kell írni, hogy mit zár le.

## Node-ok és egyebek jelölése

* X: IONodes
* IN: normalINodes
* CIN: controlINodes
* OUT: normalONodes
* FWOUT: forwardedONodes
* N: normalInternalNodes
* C: controlInternalNodes
* V: internalVars
* VG: globalVars
* CT: component as parameter
* P: params
* R: rail, sín, fix feszültségű, föld/táp jellegű csomópont
* GND: GND (nem számozható, arra az R szolgál)
* NONE: csak normál ONode esetén, nem kapcsolódik rá semmi
* F: function, csak a node számok és paraméter számok után lehet

## Paraméterek

Lehet kevesebb paramétert megadni, mint ahányat az adott alkatrész megkíván. Az elhagyott paraméterek 0 értékűek lesznek.

Áramforrásnál a paramétereket lehet nevesíteni: DC0=0 DC=0.01 AC=0.01 Phi=0 MUL=0.035. A MUL csak az I2-nél van. Ezek közül is bármit elhagyhatunk, amit elhagytunk, 0 értékű lesz. A nevesített paraméterek tetszőleges sorrendben megadhatók.

Nevesített és nevesítetlen paraméterek keverésének következménye nem definiált, ne keverjük.

## Komponens mint paraméter

Komponens, kontroller és függvény kaphat komponenst paraméterként. Felhasználni jelenleg csak függvényben lehet.

.FUNCTION blue\_ray\_section CT=1 P=3 V=3  
5 \_LOAD V3 CTS0.N0  
10 \_LOAD V2 CT0.X1  
.END FUNCTION blue\_ray\_section

.FUNCTION blue\_ray\_1 CT=3 P=1 V=2  
10 blue\_ray\_section CT0 V0 2.3e-8 P0  
20 blue\_ray\_section CT1 V1 4.7e-8 V0  
30 blue\_ray\_section CT2 RET 1.5e-7 V1  
.END FUNCTION blue\_ray\_1

.MODEL bRay1 CONTROLLER CIN=1 OUT=1 CT=3  
FUNCTION blue\_ray\_1 CT0 CT1 CT2 // no params here  
LOAD X CIN0  
STORE OUT X  
.END MODEL bRay1

.MODEL circ2 SUBCIRCUIT N=24 C=2  
Cell\_0 MODEL cell\_0 N0 N3 C0  
Cell\_1 MODEL cell\_2 N0 N1 N4  
Cell\_2 MODEL \_THIS N1 N2 N5  
Ray1 MODEL bRay1 C0 C1 Cell\_0 Cell\_1 Cell\_2  
.END MODEL circ2

I2 FCI IN=0 CIN=0 P=1 CT=1 F=square(CT0 P1) R1 N1 GND 0.5 0.1

Híváskor valamelyik tartalmazott komponens nevét kell megadni, de saját magát is megadhatja \_THIS néven.

Felhasználáskor a függvény a komponens valamely node-ját vagy változóját tudja használni. Minden komponens esetében használhatók a komponens külső node-jai, ebben az esetben CT0.X2 módon adhatjuk meg a komponenst és a node-ot. Ha egy subcircuit belső node-ját akarjuk elérni, akkor CT0.N1. Ha egy nem subcircuit típusú komponenst adunk meg, akkor a HexMG elszáll. Akkor is, ha olyan belső node-ot akarunk elérni, amilyen amúgy van az adott komponensnek.

Kontrollert is átadhatunk komponensként, melynek a külső node-jait és belső változóit CT-ként kiolvashatjuk. (A tárolt függvény változóihoz nem férhetünk hozzá.)

## .RAILS

.RAILS=4 R3=25 R2=-1.5   
.RAILS R1=0

A rail-eket (GND-ket) definiálja.

Először meg kell adni a rail-ek számát, ez a .RAILS=2 módon történik. Többször is megadható. Ha a szám kisebb, mint az előző, akkor a szimulátorban nem történik semmi, de innentől fogva nem lehet ennél nagyobb indexű railt beállítani. A biztonság kedvéért a rail-ek száma maximálva van, jelenleg 1'048'576-ra.

A felhasználói problémák elkerülése érdekében a megadottnál 1-gyel több rail van, vagyis .RAILS=N esetében R0-tól RN-ig használható.

A .RAILS sorban lehet megadni a rail értékét is R + rail index = value formában. A value támogatja a spice számformátumokat, pl. 24k. Az R0 nem adható meg, az mindig 0.

## .PROBE

Típusok:

* V: a felsorolt node-ok feszültségei külön-külön kerülnek mentésre. Lehet node helyett 0-t írni, ebben az esetben az adott helyen 0 kerül mentésre.
* I: a felsorolt node-ok áramai összegezve kerülnek mentésre. Csak a beépített komponensek normál IO node-jain van áram számítva, minden más node-on 0.
* VSUM/ISUM: a felsorolt node-ok feszültségeinek/áramainak összege, azaz probe-onként 1 érték kerül mentésre
* VAVERAGE/IAVERAGE: a felsorolt node-ok feszültségeinek/áramainak átlaga, azaz probe-onként 1 érték kerül mentésre

.PROBE P2 I circ1 Cell\_12.R1.X0 Cell\_12.R2.X0

<probe neve> <probe típusa> <full circuit neve> <node-ok>

A probe neve ismétlődhet. A probe típusa és a full circuit ebben az esetben meg kell egyezzen a korábbi definícióban megadottal. Az így felsorolt node-ok hozzácsatolódnak a korábban megadott node-listához. => hogy sok node esetén ne legyenek hosszú sorok a fájlban

Belső komponens node-ját is megadhatjuk a komponens.komponens.node sémával. Így bármilyen komponens mélységet meg tudunk adni. Normál esetben max. két szint szükséges:

* .PROBE P1 V circ1 N1848 => a full circuit 1848-as (belső) csomópontja
* .PROBE P2 V circ1 CELL24517.N1 => a full circuit CELL24517-es nevű komponensének 1-es belső csomópontja
* .PROBE P3 I circ1 CELL24517.R4.X1 => ugyanennek a cellának az R4-es komponensének az 1-es külső csomópontjához tartozó árama

## .SET

.SET VG5 5750  
.SET circ1.Cell\_12.VI1 1416.6666

Beállítható bármely globális vagy lokális változó. Lokális változó esetén a full circuit-et is megadjuk a láncban.

## .MODEL

I2 FCI IN=1 CIN=0 P=1 CT=0 F=square(IN0 P1) N1 GND N0 0.5 0.1

* A függvényvezérelt alkatrészek a típusazonosító után az adott típusú bemeneteik **extra** mennyiségét tartalmazzák. Azaz az adott alkatrésztípus normál mennyiségén felüli bemenetek számát. Jelenleg csak IN, CIN, P és CT adható meg. Ha valamelyik 0, akkor az elhagyható. Az FCI alapból 2 N bemenettel, 0 IN bemenettel, 0 CIN bemenettel és 1 P bemenettel (=G) rendelkezik.
* A fenti példában a függvény kap egy extra paraméter értéket, így a függvény IN02×P1 értéket ad, ahol P1=0.1. A FCI belső ellenállása 0.5 S vezetésű.
* A paraméterek bármilyen típusú alkatrésznél elhagyhatók, ebben az esetben 0 az értékük (ha elhagyunk, akkor mindig a végéről hagyunk el, pl. a példában ha csak egy paramétert adnánk meg, akkor az a P0 lenne.)

I0 I N0 GND 500'000 500'000 0 0 // DC=500'000 DC0=500'000

* Áramgenerátornál nevesíthetjük is a paramétereket: DC, DC0, AC, PHI, MUL lehet (MUL csak az I2 típusnál, az I-nél nem).
* Belül az áramgenerátor paraméterei DC0, DC, AC, PHI [, MUL] sorrendben vannak, de a fájlban, ha névtelenül vannak felsorolva az adatok, akkor DC, DC0, AC, PHI [, MUL] az adatok sorrendje! (Ha csak egy adatot adunk meg, az a normál DC áram legyen.)

### Kontroller:

.MODEL hiszter CONTROLLER CIN=1 OUT=1 P=3 // CIN0=T, OUT0=H, P0=T1, P1=T2, P2=W  
INIT OUT0 0 // only constant, reset sets this for value and stepstart, OUT and V can be initialized  
FUNCTION hisztercore  
LOAD X CIN0 P0 P1 P2 OUT0.STEPSTART // first is ret, the others are the pars  
STORE OUT0 X X X X X // first is ret, the others are the pars  
.END MODEL hiszter

.MODEL bRay1 CONTROLLER CIN=1 OUT=1 CT=3  
FUNCTION blue\_ray\_1 CT0 CT1 CT2 // no params here  
LOAD X CIN0  
STORE OUT X  
.END MODEL bRay1

* Egy INIT sorban csak egy node-érték pár lehet. Ez lesz az adott node kezdőértéke.
* A FUNCTION meg kell előzze a LOAD/STORE sorokat, pontosan 1 FUNCTION sor van
* A komponens paramétereket a FUNCTION sorban kell megadni, a normál paramétereket nem
* A LOAD sorban a függvény minden paraméterére meg kell adni, hogy honnan kell betölteni. Ha valamit nem akarunk betölteni, akkor X-szel jelezzük. Az első függvényparaméter a visszatérési érték. Az utolsó valahány paramétert nem kell explicite megadni, ha amúgy X-et írnánk ide.
* A STORE azt jelenti, hogy a függvény lefutása után a függvény paramétersorából hová mentse el az értéket a kontroller. A kontroller vezérlő kimeneteit itt tudjuk beállítani.
* Ha a függvény valamely paraméterénél X van, akkor annak értéke megőrződik a közvetkező futtatásig, de ezek visszalépéskor sem íródnak felül stepstarttal. Belső változó használatával ez megoldható: LOAD-dal betöltjük egy belső változóból, majd a STOR-ral elmentjük.

## .RUN

.RUN circ1 DC INITIAL ITER ITERS=5 PRE ERR=0.0002   
.RUN circ1 TIMESTEP T=10 DT=1e-6 ITER ITERS=5 PRE ERR=0.0002 INITIAL   
.RUN circ1 AC F=1000   
.RUN circ1 TIMECONST F=1000 TAU=0.2 SPD=10

* DC
  + INITIAL: a generátorok initial (DC0) értékét veszi figyelembe, ami alapból 0. A rail-ek mindig a megadott értéken vannak, initial analízisnél is.
  + Nincs INITIAL: ez a final, amikor a generátorok normál értékét vesszük figyelembe (DC). tranziens után is lehet, akkor a kiinduló érték a tranziens aktuális állapota
  + ITER: 1 DC iteráció. Ha lineáris a hálózat, akkor ez a helyes végeredmény. Ha nemlineáris, akkor csak egy lépés
  + ITERS: megadott számú DC iteráció
  + PRE: nem Newton-Raphson, hanem szukcesszív approximációs lépés
  + ERR: csak akkor van értelme, ha nincs ITER vagy ITERS. Ebben az esetben teljes DC számítás zajlik: addig ismétlődnek a DC lépések, amíg a relatív hiba a megadott érték alá nem csökken. Default értéke 0.0001 (azaz 0.01%)
* TIMESTEP
  + Minden van, ami DC-ben. Az első TIMESTEP t=0-nál indul, minden további timestep az előző után. Ha két TIMESTEP közé AC-(ke)t és/vagy TIMECONST-o(ka)t iktatunk, az semmilyen hatással nincs a következő TIMESTEP-re. Ha DC-t iktatunk közéjük, akkor t=0-ról indul újra az időszámítás és a DC-ben kapott feszültségekből. INITIAL esetén elvégez egy INITIAL DC-t a timestep előtt.
  + T: az aktuális időpontról erre az időpontra ugrik előre. Ha t<az aktuális időpont, akkor nem csinál semmit.
  + DT: ennyi idővel ugrunk előre az aktuálishoz képest.
* AC: mindig egy lépés: a korábbi DC/TIMESTEP által meghatározott munkapontban kisjelű AC számítás
  + F: frekvencia Hz-ben (elfogadja a SPICE számkonvenciókat, pl. f=10MEGHZ)
  + Használható a TAU is, akkor a frekvencia f=1/2πTAU.
* TIMECONST: az időállandó spektrum adott frekvenciához tartozó szelete
  + vagy az F frekvenciát, vagy a TAU időállandót adjuk meg. TAU = 1/2πf.
  + SPD: step per dekád, kell a számításhoz

## .SAVE

.SAVE [RAW] [APPEND] [MAXRESULTSPERROW=100] FILE=”probe.res” P1 P2

* RAW: nem ír fejlécet, a komplex értékek valós és képzetes részét tabulátorral írja ki. Ha nincs, minden mentéskor ír fejlécet, a komplex értékeket 3+4i formátumban írja ki.
* APPEND: meglévő fájl végéhez fűzi az új eredményt, ha nincs, felülírja a fájlt, ha van.
* MAXRESULTSPERROW: hány node érték után írjon ki sortörést (a komplex szám is egynek számít), default = 100.
* FILE: a fájl neve, útvonalat is tartalmazhat. Ha nincs útvonal, akkor oda teszi, ahol a hexmg\_core fut, nem a projekt mappájába.
* P1 P2: a kiírandó probe-ok, bármennyi lehet (egy probe akár többször is, ha ez valamiért kéne)

## .FUNCTION

Minden sor címkével kezdődik, ezt szövegnek tekinti a program, tehát nem csak szám lehet, és nem számít a szám értéke. Egy függvényen belül minden címke egyedi kell legyen.

.FUNCTION square P=2 V=1  
10 \_MUL V0 P0 P0  
20 \_MUL RET V0 P1  
.END FUNCTION square

src lehet: P => Parameter, V => Variable, F => Former equation

20 \_MUL RET FRET P1 => FRET: az előző sorban lévő függvényhívás RET mezője (ha ez az első, akkor a tároló függvényé)

Beépített függvények

Konstans:

* \_CONST dest value => dest = value
* \_C\_PI dest => dest = PI (3.1415…)
* \_C\_2PI dest => dest = 2\*PI (6.28…)
* \_C\_PI2 dest => dest = PI/2 (1.57…)
* \_C\_E dest => dest = e (2.71…)
* \_C\_T0 dest => dest = 273.15
* \_C\_K dest => dest = 1.38…e-23
* \_C\_Q dest => dest = 1.6…e-19

Aritmetikai:

* \_ADD dest src1 src2 => dest = src1 + src2
* \_SUB dest src1 src2 => dest = src1 - src2
* \_MUL dest src1 src2 => dest = src1 \* src2
* \_DIV dest src1 src2 => dest = src1 / src2
* \_IDIV dest src1 src2 => dest = trunc(src1 / src2)
* \_TRUNC dest src1 => dest = trunc(src)
* \_ROUND dest src1 => dest = round(src)
* \_CEIL dest src1 => dest = ceil(src)
* \_FLOOR dest src1 => dest = floor(src)
* \_MOD dest src1 src2 => dest = src1 − trunc(src1 / src2) \* src2
* \_ADDC dest src value => dest = src + value
* \_SUBC dest src value => dest = src - value
* \_MULC dest src value => dest = src \* value
* \_DIVC dest src value => dest = src / value
* \_IDIVC dest src value => dest = trunc(src / value)
* \_MODC dest src value => dest = src − trunc(src / value) \* value
* \_CADD dest value src => dest = value + src
* \_CSUB dest value src => dest = value - src
* \_CMUL dest value src => dest = value \* src
* \_CDIV dest value src => dest = value / src
* \_CIDIV dest value src => dest = trunc(value / src)
* \_CMOD dest value src2 => dest = value − trunc(value / src) \* src
* \_NEG dest src => dest = -src
* \_INV dest src => dest = 1.0 / src
* \_SQRT dest src => dest = sqrt(src)
* \_POW dest src1 src2 => dest = src1^src2
* \_POWC dest src value => dest = src^value
* \_CPOW dest value src => dest = value^src
* \_EXP dest src => dest = exp(src)
* \_NEXP dest src => dest = exp(-src)
* \_IEXP dest src => dest = exp(1/src)
* \_INEXP dest src = \_NIEXP dest src => dest = exp(-1/src)
* \_LN dest src => dest = ln src
* \_LOG dest src1 src2 => dest = logsrc1src2
* \_CLOG dest value src => dest = logvaluesrc
* \_ABS dest src => dest = |src|
* \_ASIN dest src => dest = asin(src)
* \_ACOS dest src => dest = acos(src)
* \_ATAN dest src => dest = atan(src)
* \_ASINH dest src => dest = asinh(src)
* \_ACOSH dest src => dest = acosh(src)
* \_ATANH dest src => dest = atanh(src)
* \_SIN dest src => dest = sin(src)
* \_COS dest src => dest = cos(src)
* \_TAN dest src => dest = tan(src)
* \_SINH dest src => dest = sinh(src)
* \_COSH dest src => dest = cosh(src)
* \_TANH dest src => dest = tanh(src)
* \_RATIO dest src1 src2 src3 => dest = (1 - src1) \* src2 + src1 \* src3
* \_PWL dest src x1 y1 x2 y2… => dest = pwl(src)
* \_DERIV dest src1 src2 src3 => dest = (src1 - src2) / src3
* \_DERIVC dest src1 src2 value => dest = (src1 - src2) / value
* \_VLENGTH2 dest src1 src2 => dest = sqrt(src1 \* src1 + src2 \* src2)
* \_VLENGTH3 dest src1 src2 src3 => dest = sqrt(src1 \* src1 + src2 \* src2 + src3 \* src3)
* \_DISTANCE2 dest src1 src2 src3 src4 => dest = sqrt((src1 - src3)\*(src1 - src3) + (src2 - src4)\*(src2 - src4))
* \_DISTANCE3 dest src1 src2 src3 src4 src5 src6 => dest = sqrt((src1 - src4)\*(src1 - src4) + (src2 - src5)\*(src2 - src5) + (src3 - src6)\*(src3 - src6))

Logikai:

* \_GT dest src1 src2 => dest = src1 > src2
* \_ST dest src1 src2 => dest = src1 < src2
* \_GE dest src1 src2 => dest = src1 >= src2
* \_SE dest src1 src2 => dest = src1 <= src2
* \_EQ dest src1 src2 => dest = src1 == src2
* \_NEQ dest src1 src2 => dest = src1 != src2
* \_GT0 dest src => dest = src > 0
* \_ST0 dest src => dest = src < 0
* \_GE0 dest src => dest = src >= 0
* \_SE0 dest src => dest = src <= 0
* \_EQ0 dest src => dest = src == 0
* \_NEQ0 dest src => dest = src != 0
* \_AND dest src1 src2 => dest = src1 && src2
* \_OR dest src1 src2 => dest = src1 || src2
* \_NOT dest src => dest = !src

Ugró:

* \_JMP label => jump to label
* \_JGT label src1 src2 => jump to label if src1 > src2
* \_JST label src1 src2 => jump to label if src1 < src2
* \_JGE label src1 src2 => jump to label if src1 >= src2
* \_JSE label src1 src2 => jump to label if src1 <= src2
* \_JEQ label src1 src2 => jump to label if src1 == src2
* \_JNEQ label src1 src2 => jump to label if src1 != src2
* \_JGT0 label src => jump to label if src > 0
* \_JST0 label src => jump to label if src < 0
* \_JGE0 label src => jump to label if src >= 0
* \_JSE0 label src => jump to label if src <= 0
* \_JEQ0 label src => jump to label if src == 0
* \_JNEQ0 label src => jump to label if src != 0
* \_RETURN => return
* \_RGT src1 src2 => return if src1 > src2
* \_RST src1 src2 => return if src1 < src2
* \_RGE src1 src2 => return if src1 >= src2
* \_RSE src1 src2 => return if src1 <= src2
* \_REQ src1 src2 => return if src1 == src2
* \_RNEQ src1 src2 => return if src1 != src2
* \_RGT0 src => return if src > 0
* \_RST0 src => return if src < 0
* \_RGE0 src => return if src >= 0
* \_RSE0 src => return if src <= 0
* \_REQ0 src => return if src == 0
* \_RNEQ0 src => return if src != 0

Másoló:

* \_CPY dest src => dest = src
* \_CGT dest src1 src2 src3 => src1 > src2 dest = src3
* \_CST dest src1 src2 src3 => src1 < src2 dest = src3
* \_CGE dest src1 src2 src3 => src1 >= src2 dest = src3
* \_CSE dest src1 src2 src3 => src1 <= src2 dest = src3
* \_CEQ dest src1 src2 src3 => src1 == src2 dest = src3
* \_CNEQ dest src1 src2 src3 => src1 != src2 dest = src3
* \_CGT0 dest src1 src2 => src1 > 0 dest = src2
* \_CST0 dest src1 src2 => src1 < 0 dest = src2
* \_CGE0 dest src1 src2 => src1 >= 0 dest = src2
* \_CSE0 dest src1 src2 => src1 <= 0 dest = src2
* \_CEQ0 dest src1 src2 => src1 == 0 dest = src2
* \_CNEQ0 dest src1 src2 => src1 != 0 dest = src2
* \_TGT dest src1 src2 src3 src4 => dest = src1 > src2 ? src3 : src4
* \_TST dest src1 src2 src3 src4 => dest = src1 < src2 ? src3 : src4
* \_TGE dest src1 src2 src3 src4 => dest = src1 >= src2 ? src3 : src4
* \_TSE dest src1 src2 src3 src4 => dest = src1 <= src2 ? src3 : src4
* \_TEQ dest src1 src2 src3 src4 => dest = src1 == src2 ? src3 : src4
* \_TNEQ dest src1 src2 src3 src4 => dest = src1 != src2 ? src3 : src4
* \_TGT0 dest src1 src2 src3 => dest = src1 > 0 ? src2 : src3
* \_TST0 dest src1 src2 src3 => dest = src1 < 0 ? src2 : src3
* \_TGE0 dest src1 src2 src3 => dest = src1 >= 0 ? src2 : src3
* \_TSE0 dest src1 src2 src3 => dest = src1 <= 0 ? src2 : src3
* \_TEQ0 dest src1 src2 src3 => dest = src1 == 0 ? src2 : src3
* \_TNEQ0 dest src1 src2 src3 => dest = src1 != 0 ? src2 : src3
* \_CGTC dest src1 src2 value => src1 > src2 dest = value
* \_CSTC dest src1 src2 value => src1 < src2 dest = value
* \_CGEC dest src1 src2 value => src1 >= src2 dest = value
* \_CSEC dest src1 src2 value => src1 <= src2 dest = value
* \_CEQC dest src1 src2 value => src1 == src2 dest = value
* \_CNEQC dest src1 src2 value => src1 != src2 dest = value
* \_CGT0C dest src1 value => src1 > 0 dest = value
* \_CST0C dest src1 value => src1 < 0 dest = value
* \_CGE0C dest src1 value => src1 >= 0 dest = value
* \_CSE0C dest src1 value => src1 <= 0 dest = value
* \_CEQ0C dest src1 value => src1 == 0 dest = value
* \_CNEQ0C dest src1 value => src1 != 0 dest = value

Másoló return:

* \_CGTR dest src1 src2 src3 => src1 > src2 dest = src3 + return
* \_CSTR dest src1 src2 src3 => src1 < src2 dest = src3 + return
* \_CGER dest src1 src2 src3 => src1 >= src2 dest = src3 + return
* \_CSER dest src1 src2 src3 => src1 <= src2 dest = src3 + return
* \_CEQR dest src1 src2 src3 => src1 == src2 dest = src3 + return
* \_CNEQR dest src1 src2 src3 => src1 != src2 dest = src3 + return
* \_CGT0R dest src1 src2 => src1 > 0 dest = src2 + return
* \_CST0R dest src1 src2 => src1 < 0 dest = src2 + return
* \_CGE0R dest src1 src2 => src1 >= 0 dest = src2 + return
* \_CSE0R dest src1 src2 => src1 <= 0 dest = src2 + return
* \_CEQ0R dest src1 src2 => src1 == 0 dest = src2 + return
* \_CNEQ0R dest src1 src2 => src1 != 0 dest = src2 + return

Ezeknél is csak akkor return, ha igaz:

* \_TGTR dest src1 src2 src3 src4 => dest = src1 > src2 ? src3 : src4 + return
* \_TSTR dest src1 src2 src3 src4 => dest = src1 < src2 ? src3 : src4 + return
* \_TGER dest src1 src2 src3 src4 => dest = src1 >= src2 ? src3 : src4 + return
* \_TSER dest src1 src2 src3 src4 => dest = src1 <= src2 ? src3 : src4 + return
* \_TEQR dest src1 src2 src3 src4 => dest = src1 == src2 ? src3 : src4 + return
* \_TNEQR dest src1 src2 src3 src4 => dest = src1 != src2 ? src3 : src4 + return
* \_TGT0R dest src1 src2 src3 => dest = src1 > 0 ? src2 : src3 + return
* \_TST0R dest src1 src2 src3 => dest = src1 < 0 ? src2 : src3 + return
* \_TGE0R dest src1 src2 src3 => dest = src1 >= 0 ? src2 : src3 + return
* \_TSE0R dest src1 src2 src3 => dest = src1 <= 0 ? src2 : src3 + return
* \_TEQ0R dest src1 src2 src3 => dest = src1 == 0 ? src2 : src3 + return
* \_TNEQ0R dest src1 src2 src3 => dest = src1 != 0 ? src2 : src3 + return
* \_CGTCR dest src1 src2 value => src1 > src2 dest = value + return
* \_CSTCR dest src1 src2 value => src1 < src2 dest = value + return
* \_CGECR dest src1 src2 value => src1 >= src2 dest = value + return
* \_CSECR dest src1 src2 value => src1 <= src2 dest = value + return
* \_CEQCR dest src1 src2 value => src1 == src2 dest = value + return
* \_CNEQCR dest src1 src2 value => src1 != src2 dest = value + return
* \_CGT0CR dest src1 value => src1 > 0 dest = value + return
* \_CST0CR dest src1 value => src1 < 0 dest = value + return
* \_CGE0CR dest src1 value => src1 >= 0 dest = value + return
* \_CSE0CR dest src1 value => src1 <= 0 dest = value + return
* \_CEQ0CR dest src1 value => src1 == 0 dest = value + return
* \_CNEQ0CR dest src1 value => src1 != 0 dest = value + return

Speciális:

* \_UNIT dest src => dest = src > 0 ? 1 : 0
* \_UNITT dest => dest = time > 0 ? 1 : 0
* \_URAMP dest src => dest = src > 0 ? src : 0
* \_TIME dest => dest = current time of the simulation
* \_DT dest => dest = dt
* \_FREQ dest => dest = f
* \_RAIL dest src => dest = rail(src)

CT.NODE vagy VG lehet, pl. CT.X3 vagy VG5. Az I-s függvényeknél az N egy paraméter kell legyen, mint a dest meg az src. (Az index ilyenkor paraméterként adódik át.)

* \_LOAD dest CT.NODEN => dest = CT.NODE[N].getValueDC()
* \_LOADD dest CT.NODEN => dest = CT.NODE[N].getDDC()
* \_LOADI dest CT.XNODEN => dest = CT.getCurrentDC(XNODE[N])
* \_LOADSTS dest CT.NODEN => dest = CT.NODE[N].getStepStartDC()
* \_STORE CT.NODEN src => CT.NODE[N].setValueDC(src)
* \_STORED CT.NODEN src => CT.NODE[N].setDDC(src)
* \_INCD CT.NODEN src => CT.NODE[N].incDDC(src)
* \_STORESTS CT.NODEN src => CT.NODE[N].setStepStartDC(src)
* \_ILOAD dest CT.NODE N => dest = CT.NODE[N].getValueDC()
* \_LOADD dest CT.NODE N => dest = CT.NODE[N].getDDC()
* \_LOADI dest CT.XNODE N => dest = CT.getCurrentDC(XNODE)
* \_LOADSTS dest CT.NODE N => dest = CT.NODE[N].getStepStartDC()
* \_STORE CT.NODE N src => CT.NODE[N].setValueDC(src)
* \_STORED CT.NODE N src => CT.NODE[N].setDDC(src)
* \_INCD CT.NODE N src => CT.NODE[N].incDDC(src)
* \_STORESTS CT.NODE N src => CT.NODE[N].setStepStartDC(src)

Ezekkel bele lehet írni a rail-be is, de ez elrontja az egész számítást, úgyhogy inkább ne tegye a t. user.