BOHUS MIKLÓS – DR. NÉMETH GÁBOR – TRÓN TIBOR – VARRÓ LÁSZLÓ
Budapesti Műszaki Egyetem

A SUBSET szimulációs nyelv digitális rendszerek funkcionális vizsgálatára*

ETO 681.32.001.4 : 800.92 SUBSET

A nagyméretű digitális rendszerek struktúrájának és működésének leírására használatos módszerek (kapcsolási rajzok, idődiagramok, állapotdiagramok, folyamatábrák, szöveges magyarázatok) a rendszert mindig csak egy-egy vonatkozásban jellemzik. Segítségükkel a felépítés és a pontos működés megértése elég nehéz. Sok bonyodalmat okoz — különösen nemzetközi viszonylatban — a nem egységes jelölésrendszer. Napjainkban jelentős erőfeszítések történnek annak érdekében, hogy a leírások egyértelműen dokumentálják a rendszer struktúráját és működését. Igen használhatónak bizonyulnak azok a szimbolikus nyelvek, amelyek általában alkalmasak algoritmusok és logikai rendszerek leírására.

A szimbolikus nyelvtől alapvető követelményként azt várjuk el, hogy tömör és egyszerűen felhasználható legyen, egyértelműen dokumentálja a leírt algoritmust és működési elvet, továbbá adjon lehetőséget szimulációra, azaz az illető algoritmus vagy működési elv ellenőrzésére.

A SUBSET nyelv az OSZSZ—2 nyelvvel alapkoncepciójában megegyezik.

Elkészítésekor azt a célt tűztük magunk elé, hogy alkalmas legyen a vizsgált digitális rendszer működésének dokumentációszerű leírására, és tegye lehetővé a működés szimulálását.

1. A SUBSET nyelv jellemzői

A nyelv közbülső lépésnek tekinthető az alacsony és magas szintű szimulációs nyelvek között.

Alacsony szintűnek tekinthető, mert nem olyan tömör, mint a magas szintű nyelvek. Nem engedi meg összetett kifejezések használatát. Egy utasítás csak egyetlen operátort tartalmazhat, így a felhasználónak kell ilyen részletesen megfogalmazni a feladatot. Ennek megfelelően a program bizonyos mértékig kötött formátumú assembler szintű nyelvhez hasonlít.

Más vonatkozásban a nyelv a magas szintű szimulációs nyelvek jellemzőit viseli. Ezek az utasításrendszer és a hivatkozási rendszer. Az utasítások választékában olyan összetett utasítások szerepelnek, mint regiszterléptetés, kódolás, dekódolás, regisztereken végzett egyéb műveletek. A hivatkozási rendszer lehetővé teszi, hogy az utasításokban szereplő operandusok lehetnek regiszterek, regiszterek részei, rekeszei. Egy utasítással lehet például két regiszter tetszőleges bitjeit tetszőleges sorrendben összehasonlítani.

A nyelv két változó típus deklarálását teszi lehetővé: vektor operandust (regisztert) és skalár operandust. Nem engedi meg sem késleltető elemek, sem funkcionális egységek deklarációját. A deklarált változókra, ill. konstansokra a hivatkozás szimbolikus nevükkel történik. A skalár operandus értéke mindig előjel nélküli egész szám, a vektor operandus értéke pedig logikai érték (0—1).

2. A nyelv alapelemei

A nyelv alapelemei a skalárok és a vektorok. Ezek szintaktikai és szemantikai definíciója a nyelvben a következő:

⟨*skalár*⟩::=⟨*skalár változó*⟩|⟨*skalár konstans*⟩
⟨*skalár változó*⟩::=⟨*azonosító*⟩
⟨*skalár konstans*⟩::=⟨*számjegy*⟩ | ⟨*szám*⟩ ⟨*számjegy*⟩
⟨*azonosító*⟩::=⟨*betű*⟩ | ⟨*azonosító*⟩⟨*betű*⟩ | ⟨*azonosító*⟩⟨*számjegy*⟩

A számjegyek és a betűk a nyelv alapjelei. Azonosító maximálisan hat karaktert tartalmazhat. A skalár konstansok értéke vagy a skalár változók aktuális értéke csak 32 768-nál kisebb, előjel nélküli pozitív egész szám lehet. A skalárokat a programban előfordulásuk deklarálja.

⟨*vektor*⟩::=⟨*azonosító*⟩ | ⟨*azonosító*⟩ (⟨*indexlista*⟩)
⟨*indexlista*⟩::=⟨*skalár*⟩ | ⟨*skalár*⟩:⟨*skalár*⟩
⟨*indexlista*⟩, ⟨*indexlista*⟩

Az első indexlista alak egyetlen vektorelemre utal, míg a -*skalár*-:-*skalár*- alak a vektorelemek egy intervallumát jelenti.

A vektorokat a program elején deklarálni kell. Ez külön utasítással (DR) történik úgy, hogy a vektor azonosítóját és indexhatárait közöljük.

Az indexlistában szereplő skalárok aktuális értékének mindig az indexhatárok közé kell esnie, mert indexhatár-ellenőrzést a program a futás során nem végez. Amennyiben egy vektorra a programban csak azonosítójával hivatkozunk, a műveletben a vektor a deklaráló utasításban szereplő indexhatárokkal vesz részt.

Ha egy műveletben különböző hosszúságú vektor operandusok vannak, akkor a műveletekben részt vevő vektorelemek számát az eredményvektor hossza határozza meg. Ilyenkor a rövidebb operandusokat balra nullával egészítjük ki. A műveletekben az egyes operandusok jobbra igazodva vesznek részt.

pl.: *AREG* (0:7)=0, 0 , 0 , 0, 0 ,0, 0, 0,
BREG (0:3)=1, 1, 0, 1,
AREG (2:4):=*BREG* értékadó utasítás eredménye

*A munka a Számítástechnikai Koordinációs Intézet megbízásából készült.

Beérkezett: 1972. III. 4.

AREG (0:7)=0, 0, 1, 0, 1, 0, 0, 0,

AREG (4:7):=*BREG* (3:0) értékadó utasítás eredménye

AREG (0:7)=0, 0, 0, 0, 1, 0, 1, 1,

3. Az utasítások általános alakja

Az utasítások szerkezete és írásának formátuma a következő:

	1. oszlop	üres, vagy „*C*" betű,
2.—	5. oszlop	üres,
6.—	8. oszlop	címke,
	9. oszlop	üres,
10.—13.	oszlop	az utasítás mnemonikus műveleti jele,
14.—15.	oszlop	üres,
16.—72.	oszlop	operandusmező,
73.—80.	oszlop	kártyaazonosító.

Ha az első oszlop tartalma „*C*" betű, az egész kártya tartalmát commentnek tekinti a program.

A címke három számjegyű decimális egész szám lehet. A szóköz itt nullának számít.

Az utasítások mnemonikus jelét az összefoglaló táblázatban láthatjuk.

Az operandusmező tartalmazza a műveletben részt vevő operandusokat. Operandus lehet skalár vagy vektor. Minden utasítás meghatározza, hogy hány és milyen típusú operandust kell felhasználni, s ezeket milyen sorrendben kell felírni. Általában az első operandus helyén eredménynek, ugrási címkéknek vagy FORM címkének kell állnia, a műveleti jelet pedig a második és harmadik operandus közé kell képzelni.

Az operandusokat egymástól vesszővel kell elválasztani. Az operandusmezőben a betűköz értéktelen, figyelmen kívül marad. Ha egy utasítás nem fér ki egy kártyára, tetszőleges számú folytatáskártya iktatható be. Egy kártya 72. oszlopában álló betűköztől különböző karakter jelzi, hogy folytatáskártya következik. Ez a karakter része az utasításnak, és a folytatáskártyát a 16. oszlopban kell kezdeni.

4. Utasítások

Vektor deklaráció

Nem végrehajtandó utasítás. A fordítóprogrammal közli a vektorok nevét, indexhatárait, esetleges kezdeti tartalmát. A deklaráló utasításnak vagy utasításoknak mindig az első végrehajtandó utasítás előtt kell állniuk. Az utasítás operandusa a vektorlista.

⟨*vektorlista*⟩::=⟨*azonosító*⟩ | ⟨*azonosító*⟩|(⟨*kezdőindex*⟩:⟨*végindex*⟩) | ⟨*vektorlista*⟩ / ⟨*pozíció*⟩// | ⟨*vektorlista*⟩, ⟨*vektorlista*⟩

⟨*kezdőindex*⟩::=⟨*szám*⟩

⟨*végindex*⟩ ::=⟨*szám*⟩

⟨*pozíció*⟩ ::=⟨*szám*⟩ | ⟨*szám*⟩:⟨*szám*⟩ | ⟨*pozíció*⟩, ⟨*pozíció*⟩

A vektorlista harmadik alakját használjuk fel arra, hogy közöljük azokat a pozíciókat (vektorindexeket), amelyek kezdeti értékét 1-re akarjuk állítani. A ⟨*szám*⟩:⟨*szám*⟩ alakkal intervallumot adhatunk meg. A kezdeti értékkel el nem látott vektorokat vagy a a pozíció-val meg nem jelölt elemeket a fordítóprogram nullával tölti fel.

Pl.: *SPECV* (0:7) (0:3,6)=1, 1, 1, 1, 0, 0, 1, 0.

Értékadás

Az értékadás művelete egyaránt értelmezett skalár és vektor operandusokra. Kizárólag skalár-skalár, vektor-vektor operanduspár van megengedve.

Léptetés

Ebbe a csoportba tartozó műveletek vektoroperandusra vonatkoznak. A léptetés típusát a különböző műveleti jelek specifikálják, s a lépésszámot mint skalárt a második operandus helyén adjuk meg. Az eredmény a léptetendő vektorban marad. Ötféle léptetési típus van megengedve, mindegyik jobbra is és balra is. Ezek a következők:

a) a kiürülő pozíciók változatlanok,
b) a kiürülő pozíciókba 0 lép be,
c) a kiürülő pozíciókba 1 lép be,
d) a kiürülő pozícióba a szélső lép be,
e) a ciklikus léptetés, a kiürülő pozícióba a kilépő lép be.

Pl.: *AREG* (0:7)=0, 1, 0, 0, 1, 1, 0, 1,

hármat léptetünk balra, s az eredmények a következők:

a) *AREG* (0:7)=0, 1, 1, 0, 1, 1, 0, 1,
b) *AREG* (0:7)=0, 1, 1, 0, 1, 0, 0, 0,
c) *AREG* (0:7)=0, 1, 1, 0, 1, 1, 1, 1,
d) *AREG* (0:7)=0, 1, 1, 0, 1, 1, 1, 1,
e) *AREG* ⟨0:7)=0, 1, 1, 0, 1, 0, 1, 0,

Aritmetikai műveletek

A négy aritmetikai alapművelet van megengedve skalár operandusokkal. Az osztás egész típusú.

Logikai műveletek

A logikai műveletek vektoroperandusokra vannak értelmezve, és a vektor elemeire bitenként vonatkoznak. A következő műveleteket valósítottuk meg: negálás, és, vagy, kisebb, kisebb vagy egyenlő, egyenlő, nagyobb vagy egyenlő, nagyobb, nem egyenlő.

Aritmetikai relációk

Ezek az utasítások két skalár operandusra vannak értelmezve. A reláció teljesülése esetén a vezérlés az első operandus helyén feltüntetett címkére adódik át, egyébként a következő utasítás hajtódik végre. A következő relációk vannak: $<$, $\leq$, $=$, $\geq$, $>$, $\neq$

Logikai feltételes vezérlésátadás

Egyetlen vektorelem operandust használ fel. Ha ennek a tartalma 1, akkor a vezérlés átadódik az

utasításban szereplő címkére, egyébként a következő utasítás hajtódik végre.

Feltétel nélküli vezérlésátadás

Átadja a műveletek végrehajtási sorrendjét az utasításban szereplő címkére.

Maradék előállítása

A műveletet skalár operandusok között értelmezzük. Eredményül az operandusokon végzett egész osztás maradékát kapjuk.

Értékmeghatározás

A művelet eredménye skalár, amelyet egy vektor operandusból nyerünk úgy, hogy a vektor tartalmát különböző számrendszerben ábrázolt számnak tekintjük.

Pl.: *AREG* (0:7)=0, 0, 1, 0, 1, 0, 0, 1,

Ha ez a szám bináris, oktális vagy hexadecimális, akkor értéke 41_{10}. Amennyiben decimális szám (BCD), akkor értéke 29_{10}. Ezt az értéket veszi fel az operandusmezőben szereplő skalár.

Dekódolás

Két különböző hosszúságú vektor között értelmezett művelet. A művelet eredményvektorának minden helyértéke 0 lesz, kivéve azt, amelyik elem sorszámát (nem indexét) az operandus vektor bináris tartalma megjelöli. Az eredményvektor feltöltésekor jobbra illesztés történik.

Pl.: *AREG* (0:3)≡0, 1, 0, 1.

A művelet eredménye az operandustól függő:

Operandus	Eredmény
BREG (0:7)	*BREG* (0:7)≡0, 0, 1, 0, 0, 0, 0, 0,
BREG (0:5)	*BREG* (0:7)≡1, 0, 0, 0, 0, 0, 0, 0,
BREG (5:0)	*BREG* (0:7)≡0 ,0, 0, 0, 0, 1, 0, 0.

Kódolás

A művelet az értékmeghatározás ellentettje, skalár operandusból vektor eredményt állít elő. A vektor tartalma különböző számrendszerbeli szám lehet (2, 8, 10, 16).

Pl.: *SKALAR*=50

bináris	*AREG* (0:7)=0, 0, 1, 1, 0, 0, 1, 0,
decimális (BCD)	*AREG* (0:7)=0, 1, 0, 1, 0, 0, 0, 0.

Input-output műveletek

Az input-output utasítások egy vagy több különböző formátumú beolvasását, illetve egy vagy több sor különböző formátumú kiírását teszik lehetővé. A beolvasás az input-lista elemeibe, a kiírás az output lista elemeiből történik a megfelelő formátum szerint.

A két lista a következő lehet:

⟨*I/0 lista*⟩::=⟨*skalár*⟩ | ⟨*vektor*⟩|⟨*I/0 lista*⟩,|⟨*I/0 lista*⟩.

Az input-output utasítás tartalmazza a vonatkozó FORM utasítás címkéjét. A FORM utasítás határozza meg a kiírás vagy beolvasás formátumát a zárójelek közé tett FORM lista segítségével.

⟨*FORM lista*⟩::=⟨*FORM listaelem*⟩, ⟨*FORM lista*⟩
⟨*FORM listaelem*⟩::=⟨*ismétlési tényező*⟩*B*⟨*mezőhosszúság*⟩
⟨*ismétlési tényező*⟩*S*⟨*mezőhosszúság*⟩
⟨*ismétlési tényező*⟩*A*⟨*mezőhosszúság*⟩
⟨*ismétlési tényező*⟩*X*⟨*betűköz szám*⟩
⟨*ismétlési tényező*⟩*L*⟨*soremelés szám*⟩
⟨*mezőhosszúság*⟩*H*⟨*mezőtartalom*⟩

⟨*ismétlési tényező*⟩::=⟨*szám*⟩
⟨*mezőhosszúság*⟩ ::=⟨*szám*⟩
⟨*soremelés szám*⟩ ::=⟨*szám*⟩
⟨*betűköz szám*⟩ ::=⟨*szám*⟩
⟨*mezőtartalom*⟩ ::=⟨*tetszőleges, a jelkészletben levő karakter*⟩.

Az ismétlési tényező, a mezőhosszúság, a betűköz szám és a soremelés szám elmaradása esetén a program 1-et tételez fel.

Az egyes lehetőségek értelmezése:

— Az első formátummal bináris értékeket (0,1) írhatunk ki vagy olvashatunk be,
— a másodikkal skalár értékeket,
— a harmadik alfanumerikus adatok beolvasását és kiírását teszi lehetővé. Két alfanumerikus adatot (karaktert) egy skalár változóban vagy egy vektorelemben lehet tárolni,
— a negyedik betűközök kihagyását teszi lehetővé kiíratásnál és beolvasásnál egyaránt,
— az ötödikkel üres sort lehet kihagyni vagy kártya leolvasását átugrani,
— a hatodik formátum a mezőhosszúság által meghatározott számú s a *H* után következő karakterek kiírását teszi lehetővé (Holerith-konstans).

END művelet

A program futásának befejezését idézi elő.

5. A SUBSET nyelven írt programok futtatása

A nyelv szabályainak megfelelően megírt programot és a szimulációhoz szükséges adatokat lyukkártyára kell lyukasztani. A program első kártyája mint címkártya a feladat azonosítására használható fel.

A program lefordításához és futtatásához egy fordító és értelmező program készült a SIEMENS 4004/45 gépre. A szimuláció két részből áll:

A fordítást, az ezt kísérő szintaktikus ellenőrzést és hibajelzést egy FORTRAN nyelven írt program végzi. A szintaktus hibák jelzése szöveges üzenetek formájában történik. Hibátlan program esetén fordítás után indul a végrehajtás. Az egyes utasításokat, tekintettel azok bonyolultságára, egy-egy assembler nyelven írt programrész hajtja végre.

Az értelmező programnak és a futtató résznek, valamint a futtatáshoz szükséges területeknek az összes memóriaszükséglete 56 kbyte. Ebben az esetben a SUBSET nyelven írt program méreteire az alábbi korlátozások érvényesek:

címkék száma	max 100,
FORM címkék száma	max 100,
skalárok száma	max 200,
vektorok száma	max 100,
vektorelemek száma	max 3000,/
az utasítások száma	max 500—600,
a FORM mezők együttes hossza	max 1000 karakter.

A fordítás sebessége 3—4 kártya/s, a végrehajtási sebesség átlagosan 100 művelet/s, de ez erősen függ az utasításoktól.

Mivel a szimuláció során egy berendezés működésének vagy egy algoritmusnak csak néhányszori szimulációjára van szükség, ez a sebesség elfogadható.

6. Alkalmazási lehetőségek

A nyelv minden olyan területen alkalmazható, ahol felmerül a szimuláció szükségessége. Általában olyan digitális berendezések fejlesztése esetén, amelyek valamilyen új rendszertechnikai megfontolást tartalmaznak, új elv vagy algoritmus szerint működnek. Ilyen esetekben van elsősorban szükség arra, hogy az áramköri részletektől eltekintve egészében lássák a rendszer működését.

Az általános szimulációs problémákon kívül sikeresen alkalmazható a nyelv megtervezett nagyberendezések megépítés előtti vizsgálatára. A működésben felmerülő hibáknak ilyen stádiumban való kimutatása jelentős idő- és anyagmegtakarítást jelent. A szükségessé váló módosítások még könnyen elvégezhetők.

Tipikus alkalmazási lehetőséget kínálnak az alábbi területek: digitális célberendezések fejlesztése, digitális műszerek, digitális szerszámgép-vezérlő rendszerek, DDC-rendszerek szabályozóberendezései, perifériák interface áramkörei, csatornavezérlő rendszerek, számítógépek adafeldolgozó egységei, adatátviteli berendezések hibajelző, hibajavító egységei stb.

1. táblázat

Az utasítások összefoglaló táblázata

Utasítás	Műv. jel	1. operandus	2. operandus	3. operandus
Vektor deklarálás	**DIM**	vektorlista		
Értékadás	:=	eredmény (skalár, vektor)	operandus (skalár, vektor)	
Léptetés				
balra, kiürülő változatlan	**LS**	operandus és eredmény (vektor)	lépésszám (skalár)	
balra, 0 lép be	**LS0**			
balra, 1 lép be	**LS1**			
balra, ciklikusan	**LSC**			
balra, szélső lép be	**LSE**			
jobbra, kiürülő változatlan	**RS**			
jobbra, 0 lép be	**RS0**			
jobbra, 1 lép be	**RS1**			
jobbra, ciklikusán	**RSC**			
jobbra, szélső lép be	**RSE**			
Aritmetikai műveletek				
összeadás	+	eredmény (skalár)	operandus (skalár) (kisebbítendő, osztandó)	operandus (skalár)
kivonás	—			
szorzás	×			
osztás	/			
Logikai műveletek				
negálás	**NOT**	eredmény (vektor)	operandus (vektor)	operandus (vektor)
és	**&**			
vagy	**V**			
Logikai reláció				
kisebb	**LTL**	eredmény (vektor)	operandus (vektor)	operandus (vektor)
kisebb vagy egyenlő	**LEL**			
egyenlő	**EQL**			
nagyobb vagy egyenlő	**GEL**			
nagyobb	**GTL**			
nem egyenlő	**NEL**			

1. táblázat folytatása

Utasítás	Müv. jel	1. operandus	2. operandus	3. operandus
Aritmetikai reláció (feltételes vezérlésátadás)		ugrási címke (skalár konst.)	operandus (skalár)	operandus (skalár)
kisebb	LT			
kisebb vagy egyenlő	LE			
egyenlő	EQ			
nagyobb vagy egyenlő	GE			
nagyobb	GT			
nem egyenlő	NE			
Logikai feltételes vezérlésátadás	IF	ugrási címke (skalár konst.)	operandus (vektorelem)	
Feltétel nélküli vezérlésátadás	JMP	ugrási címke (skalár konst.)		
Maradék előállítása	MOD	eredmény (skalár)	operandus (skalár)	operandus (skalár)
Értékmeghatározás, a vektor tartalma		eredmény (skalár)	operandus (vektor)	
bináris	SB			
bináris	S			
oktális	SO			
hexadecimális	SH			
decimális (BCD)	SD			
Dekódolás	DEC	eredmény (vektor)	operandus (vektor)	
Kódolás, a vektor tartalma:		eredmény (vektor)	operandus (skalár)	
bináris	CODB			
bináris	COD			
oktális	CODO			
hexadecimális	CODH			
decimális (BCD)	CODD			
Input	BEAD	FORM címke (skalár konst.)	input-lista (skalár, vektor)	
Output	WRIT	FORM címke (skalár konst.)	output-lista (skalár, vektor)	
Formátum meghatározása	FORM	FORM lista		
Futás vége	END	—	—	

IRODALOM

[1] *Kelly, J. J. Jr.—Lochbaum, C.—Vyssotsky, V. A.* A block-diagram compiler. Bell Syst. Techn. J. Vol. 40. No. 3 May 1961.

[2] *Gordon, G.:* A general purpose systems simulation program. IBM System J. Vol. 1. Sept. 1962.

[3] *Iverson, K. E.:* A Programming Language. John Wiley and Sons, New York, 1962.

[4] *Markowitz, H. M.—Hausner, B.—Karr, H. W.:* Simscript: A Simulation Programming Language. The RAND Corporation, RM—3310. November 1962.

[5] *Young, K.:* A User's Experience with There Simulation Languages (GPSS, SIMSCRIPT, SIMPAX). Santa Monica, California, System Development Corp. TM—1755/000/00, 1963.

[6] *Knuth, D. E.—McNeley, J. L.:* SOL-A Symbolic Language for General-Purpose Systems Simulation. IEEE Trans. Electr. Comp. Vol. EC—13 Aug. 1964.

[7] *Proctor, R. M.:* A logic design transistor experiment demonstrating relationships of language to systems and logic design. IEEE Trans. Electronic Computers, Vol. EC—13, pp. 422—430, Aug. 1964.

[8] *Schlaeppi, H. P.:* A formai language for describing machine logic, timing and sequencing (LOTIS). IEEE Trans. Electronic Computers, Vol. EC—13, 1964.

[9] *Falkoff, A. D.—Iverson, K. E.—Sassenguth, R.:* A Formai Description of System/360. IBM System J. Vol. 3. No 3. 1964.

[10] *Schorr, H.:* Computer-aided digital systems design and analysis using a register transfer language. IEEE Trans. Electronic Computers, Vol. EC—13 pp. 730—737 Dec. 1964.

[11] *Weinberg, G. M.:* PL/1 Programming Primer. McGraw Hill, New York. 1966.

[12] *Dumey, J. R.—Dietmeyer, D. L.:* A digital system design language (DDL). IEEE Trans. Computers, Vol C—17, pp. 850—861, September 1968.

[13] *Matjuhin, N. A.:* Primenenije vücsiszlityelnüh masin dija proektirovanija cifrovüh usztrojsztv. Szovjetszkoje Radio, 1968.

[14] *Chapin, N.:* 360 Programming in Assembly Language, McGraw-Hill, New York, 1968.

[15] General Purpose Systems Simulator III.

— Introduction IBM Corporation Manual No B20—0001—0.

— User's Manual IBM Corporation Manual No H20—0163—1.

[16] Duley, *J. R.—Dietmeyer, D. L.:* Translation of a DDL Digital System Specification to Boolean Equations. IEEE Trans. Computers, Vol. C—18, 1969.

[17] Dertovzos, *M. L.—Kaliski, M. E.—Polzen, K.* P.: On-line Simulation of Block-Diagram Systems. IEEE Trans. Computers, Vol. C—18, 1969.

[18] *Friedman, T. D.—Yang, S. C.:* Methods Used in an Automatic Logic Design Generator (ALERT). IEEE Trans. Computers, Vol. C—18 Jul. 1969.

[19] *Hays, G. G.:* Computer Aided Design: Simulation of Digital Design Logic. IEEE Trans. Computers. Vol. C—18, Jan. 1969.

[20] Jazik dija opiszanija sztrukturnüh algoritmov i szhem (OCC—2). Otraszlevoj sztandart: OCT 4. 10.000.025 Moszkva, 1969.

[21] *Bohus M.:* Szimbolikus nyelvek felhasználása digitális rendszerek funkcionális és parametrikus szimulációjára. Híradástechnika XXIII. évf. 6. sz.

[22] *Bohus M.—Németh G.—Trón T.—Varró L.:* Strukturális algoritmusok szimulációja (OSZSZ—2 SUBSET). Számítástechnikai Koordinációs Intézet, Budapest, 1970.

[23] *Glutkov, V. M.:* Perszpektivü avtomatizacii proektirovanija vücsiszlityelnüh masin. Vesztnik CCCP No. 4. 1967.

[24] *Herscovitch, H.—Schneider, I. H.:* GPSS III — an Expanded General Purpose Simulator. IBM Systems Journal, Vol. 4. No. 3. 1965.

[25] *Markowitz, H. M.—Hansner, B.:* Simscript — a Simulation Programming Language. Prentice-Hall, New York, 1963.

[26] *Knuth, D. E.—McNeley, J. L.:* A Formai Definition of SOL. IEEE Trans. on E. C. Vol. 5. EC—13, Aug. 1964.

[27] *Ole-Jahon-Dahl—Kristen Nygaard:* SIMULA — a Language for Programming and Description of Discrete Event Systems. Norwegian Computing Center, May 1966.

[28] *Kalinicsenko, L. A.:* Formalnoje opiszanije jazika SZLENG. c. szb „Teorija automatov" vüp. 1. izd. I. K. A. N. USZSZR Kiev, 1967.

[29] *Kalinicsenko, L. A.:* SZLENG-ekszperimentalnüj jazik programirovanija, orientirovannüj na opiszaije i modelirovanije vücsiszlityelnüh masin is szisztem. V. cb. „Teorija avtomatov" vüp. 1. izd. I. K. A. N. USZSZR 1967.

[30] *Gluskov, V. M.—Kalinicsenko, L. A.—Marjanovics, G.* P. *—Moszkalenko V. M.—Szahnjuk, M. A.:* SZLENG-Szisztema programmirovanija dija modelirovanija diszkretnüh szisztem. Kiev, 1969.

[31] *Bohus M.—Flesch I.—Géher K.—Németh G.—Pápay Zs. —Szittya O.—Theisz P.:* Logikai rendszerek számítógépes szimulációja. Tanulmány a Számítástechnikai Koordinációs Intézet számára. Budapest, 1969.