

Zadaća 2

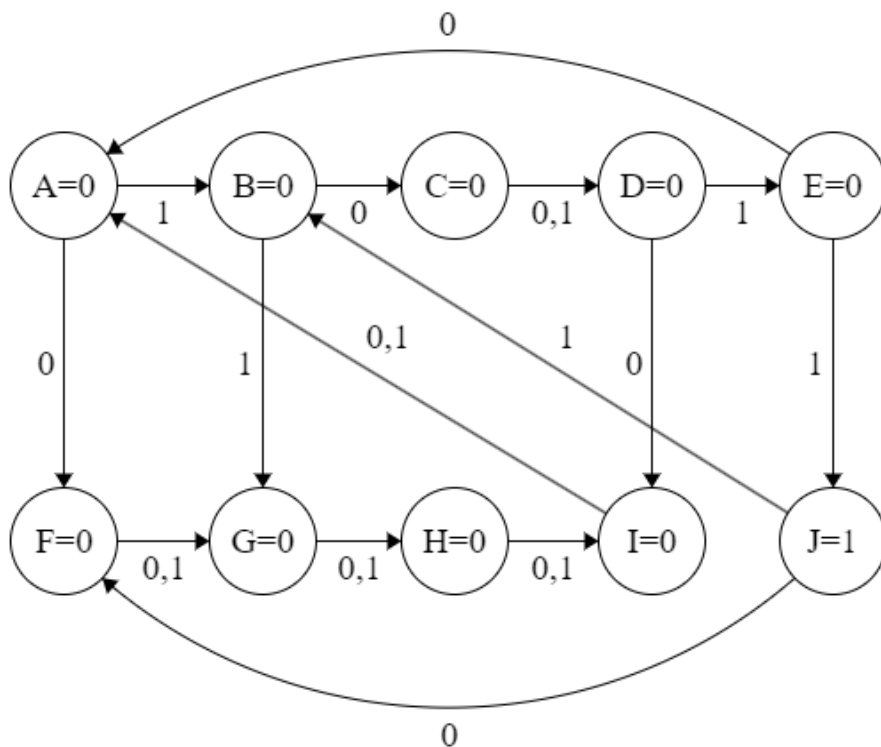
Logički dizajn

Student: Bakir Činjurević, bcinjarevi1@etf.unsa.ba

Zadatak 1: Mooreovi i Mealyevi automati

Projektovati kolo koje sinhrono očitava sekvencu bita na ulazu, a na izlazu daje 1 kada očita sekvencu "10111" ili "10011" koristeći Mooreov i Mealyev automat.

Izlaz Mooreovog automata ovisi samo o trenutnom stanju. Stanja automata grafički izgledaju ovako:



Potrebno nam je $\lceil \log_2 10 \rceil = 4$ memorijska elementa za kodiranje stanja. U tablici istine, sljedeće stanje ovisi o trenutnom stanju i ulazu. Nakon što raspišemo tablicu istine, bitovi za D flip flove $J_3J_2J_1J_0$ (iako služe za D flip flove, biti su označeni sa J ; ovo je sitnica koju sam prekasno shvatio), redom, su određeni svojim izrazima preko sljedećih K-mapu:

f		$q1,q0,u$							
		000	001	011	010	110	111	101	100
$q3,q2$	00	0	0	0	0	1	0	0	0
	01	0	1	0	0	1	1	0	0
	11	-	-	-	-	0	0	-	-
	10	0	0	-	-	-	-	-	-

f		$q1,q0,u$							
		000	001	011	010	110	111	101	100
$q3,q2$	00	1	0	1	0	1	1	0	0
	01	0	0	1	1	1	1	1	1
	11	-	-	-	-	0	0	-	-
	10	1	0	-	-	-	-	-	-

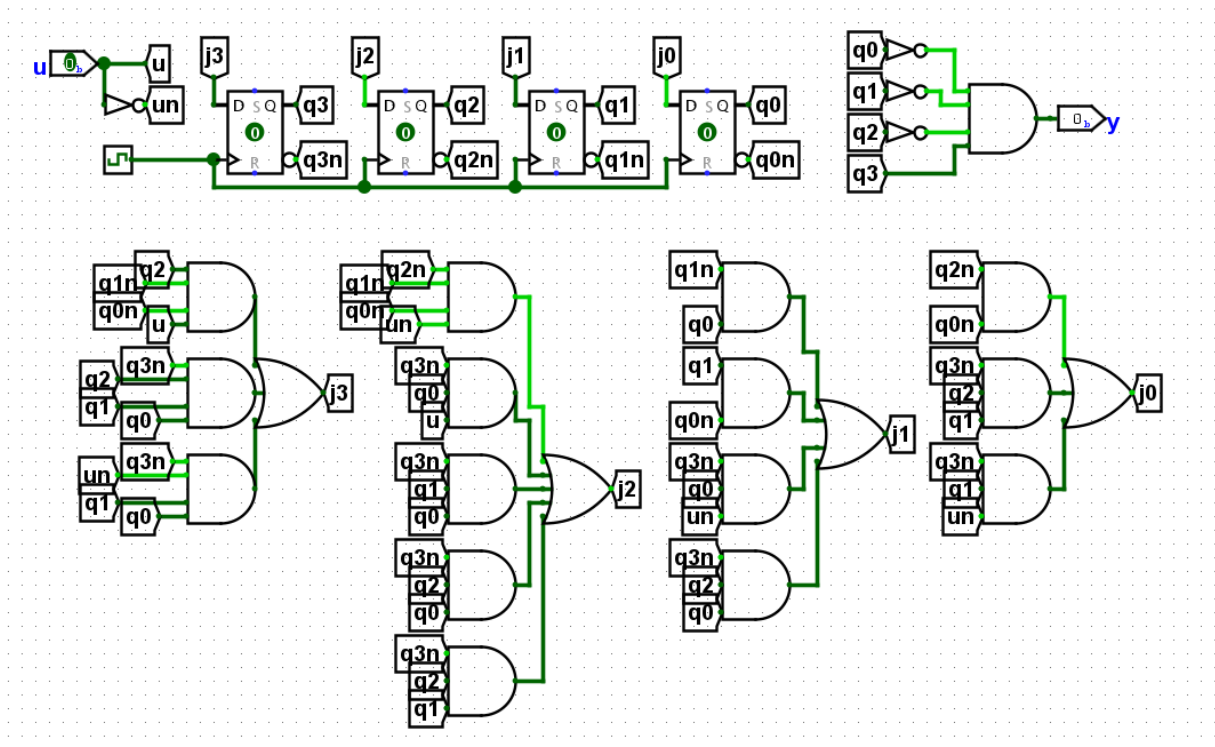
f		$q1,q0,u$							
		000	001	011	010	110	111	101	100
$q3,q2$	00	0	0	1	1	1	0	1	1
	01	0	0	1	1	1	1	1	1
	11	-	-	-	-	0	0	-	-
	10	0	0	-	-	-	-	-	-

f		$q1, q0, u$							
		000	001	011	010	110	111	101	100
$q3, q2$	00	1	1	0	0	1	0	1	1
	01	0	0	0	0	1	1	1	1
	11	-	-	-	-	0	0	-	-
	10	1	1	-	-	-	-	-	-

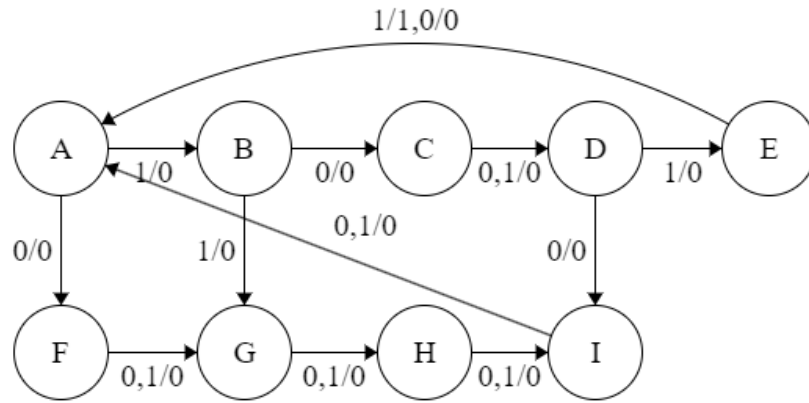
Neprečišćeni izrazi koje dobijemo su:

- $J_3 = q_2 q_1' q_0' u + q_3' q_2 q_1 q_0 + q_3' q_1 q_0 u'$
- $J_2 = q_2' q_1' q_0' u' + q_3' q_0 u + q_3' q_1 q_0 + q_3' q_2 q_0 + q_3' q_2 q_1$
- $J_1 = q_1' q_0 + q_1 q_0' + q_3' q_0 u' + q_3' q_2 q_0$
- $J_0 = q_2' q_0' + q_3' q_2 q_1 + q_3' q_1 u'$

Naravno, članovi ovih izraza se mogu grupisati radi minimizacije broja kola, ali to nije fokus trenutnog zadatka. Slika kola u Logisimu je sljedeća:



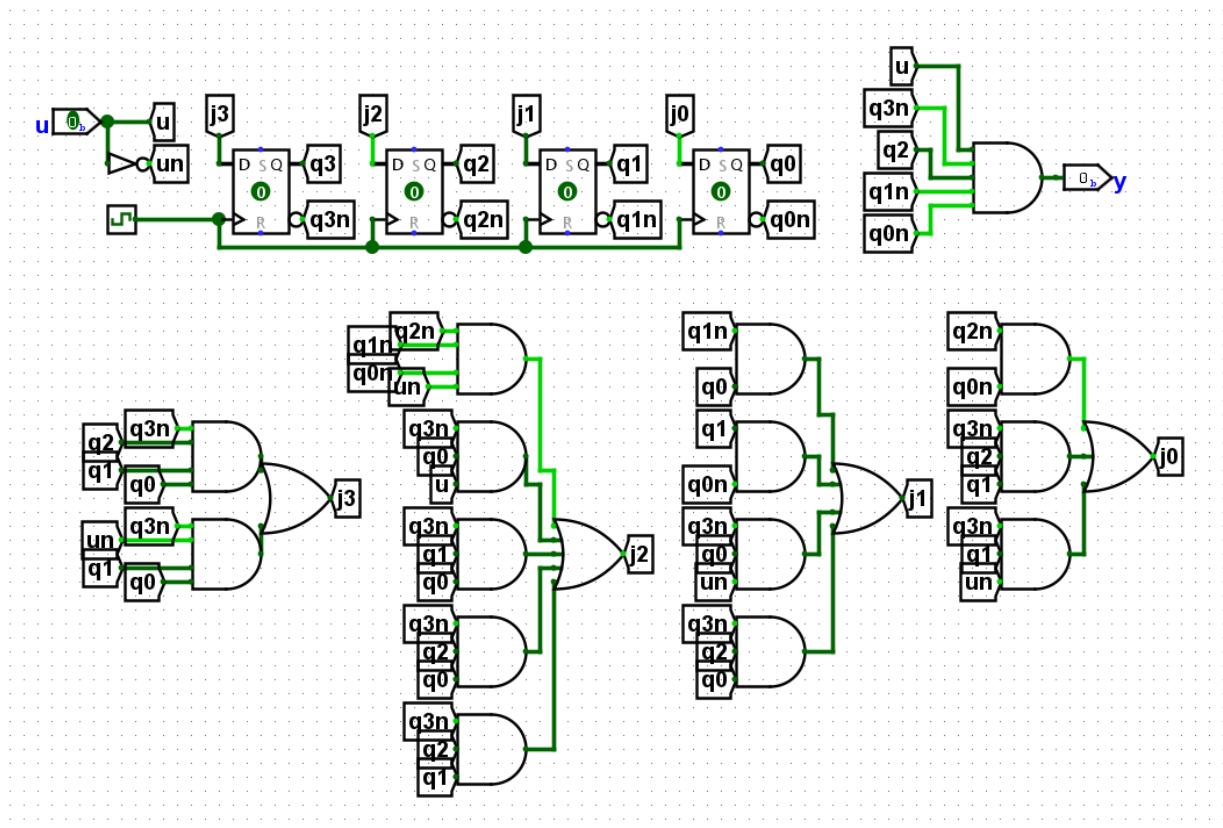
Što se tiče Mealyovog automata, njegova slika je sljedeća:



Vidimo da je vrlo sličan Mooreovom, pri čemu je jedina razlika što izlaz ovisi ne samo od trenutnog stanja, već i od ulaza u presudnom trenutku. Dakle, na kraju izvedbe Mealyevog automata na ovaj zadatak, jedino što se različi Mooreu je slijedeće:

- U izrazu za J_3 više ne figuriše minterma $q_2q'_1q'_0u$, već je izraz jednak samo $J_3 = q'_3q_2q_1q_0 + q'_3q_1q_0u'$, i
- Izlaz se dobiva izrazom $Y = uq'_3q_2q'_1q'_0$.

Šema u Logisimu izgleda ovako:



Kao što je već spomenuto, ovaj automat se može uljepšati grupisanjem članova izraza za J bitove, ali se u to nećemo upuštati ovdje.

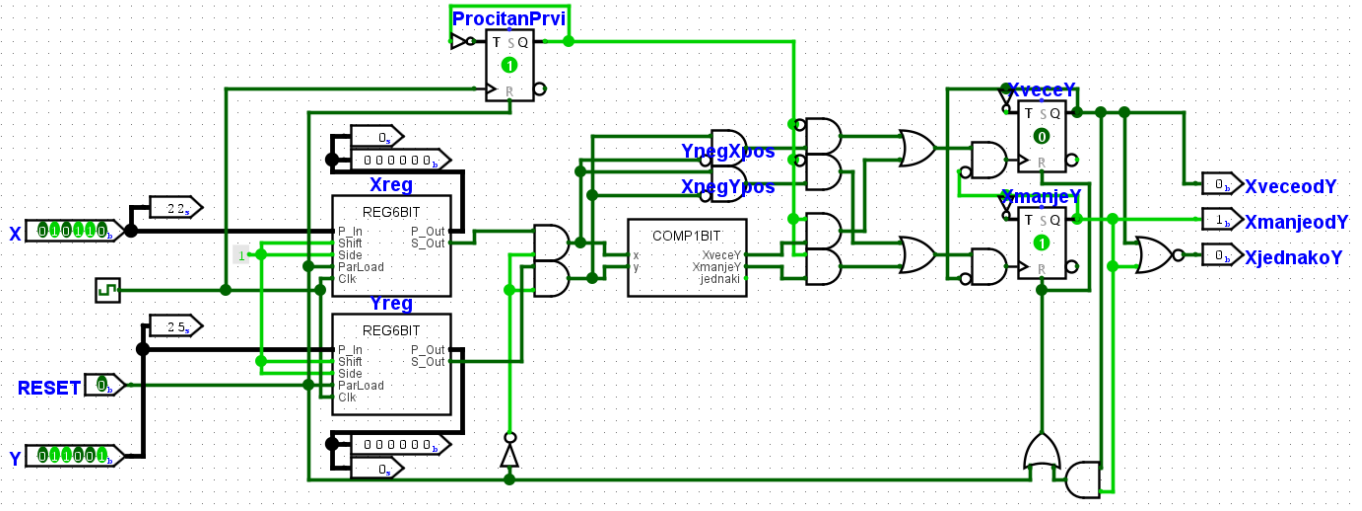
Zadatak 2: Vremensko-iterativni 6bitni komparator

Projektovati i simulirati u Logisimu 6 bitni komparator kao vremensku iterativnu strukturu.

Kolo izgleda ovako, nakon čega slijedi objašnjenje:

UPUTE:

1. postavite zeljene brojeve X i Y
2. upalite RESET i okinite sat
3. ugasite RESET i okidajte sat



Glavne ideje za izradu ovog zadatka su:

- Za potrebe vremenske iterativnosti, koristi se serijski izlaz shift registra koji ulazi u jednobitni komparator,
- Poredimo brojeve u 2kk formatu, tako da moramo prvi bit posebno tretirati,
- Kada komparator odluči da je jedan izlaz veći od drugog, postavi stanje flip flopa za taj izlaz i zadrži to stanje do kraja poređenja,
- Ako komparator nije našao veći ili manji broj, pretpostavlja se da su jednaki, tj. $(x = y) \Leftrightarrow x < y \vee x > y$

Prvi problem je tretman prvog (najvišeg) bita. Prvo imamo FF koji je jednak jedinici ako smo obradili prvi bit, a jednak nuli ako nismo. Imamo dva slučaja pri poređenju brojeva X i Y u smislu prvog bita prije nego što je obrađen: 1. X je negativan a Y je pozitivan, u kojem slučaju postavljamo krajnji rezultat da je $X < Y$, i 2. X je pozitivan a Y je negativan, gdje postavljamo krajnji rezultat $X > Y$. Napomenimo da nakon postavljanja krajnjeg rezultata $X < Y$ ili $X > Y$ dalje promjene rezultata su onemogućene.

Ako se desi da su najviši bitovi X i Y oboje nule ili jedinice, onda se ostatak postupka prepušta poređenju jednobitnog komparatora, koji tek dobija ulogu nakon što je obrađen prvi bit. Opet, ako komparator pronađe veći element, šalje signal krajnjim flip flipovima i oni se zaključavaju, onemogućujući dalje promjene.

Treba napomenuti da šestobitni registri (radi se o istom registru rađenom na tutorijalu, osim što je ovaj šestobitan) shiftaju svoj serijski izlaz ulijevo, kako bi se obrađivao najviši bit prvo. Još jedan detalj ovog kola je failsafe na kraju, koji u slučaju da se ikada pojavi da je X istovremeno veće i manje od Y resetuje krajnje flip flipove, i time vraća rezultat na jednakost.

Upravljanje ovim kolom je poprilično jednostavno. Prvo se postave bitovi X i Y, pa se postavi RESET pin na jedinicu, i nakon toga okine sat. Razlog okidanju sata je da se paralelno X i Y upišu u svoje registre. Nakon što su se X i Y smjestili u registre, što možemo vidjeti na izlaznim pinovima registara, onda se RESET treba vratiti na nulu. Neposredno nakon vraćanja RESETa na nulu, poređenje počinje, i na izlazu se vidi šta je komparator zaključio nakon što je vidio prva dva bita. Nakon ovoga, korisnik okida sat dok registri ne shiftaju sve ulijevo.

Zadatak 3: Keš memorija

Projektovati četverostruko grupno-asocijativni keš kapaciteta 2MB za 64-bitnu memoriju kapaciteta 8GB. Veličina bloka je 64 bajta.

Izlistaćemo dostupne podatke:

- $capRadno = 8GB$,
- $bRadno = 64b$,
- $nAdr = capRadno/bRadno = 8GB/64b = 8GB/8B = 1G = 2^{30}$,
- $adr = \log_2 nAdr = \log_2 2^{30} = 30$,
- $ass = 4$,
- $word = 8B$,
- $block = 64B$,
- $cap = 2MB$.

Broj linija u kešu se računa po formuli $N = \frac{cap}{block \cdot ass}$, pa će biti $N = \frac{2MB}{64B \cdot ass} = \frac{2^{21}B}{2^8B} = 2^{13}$.

Biti indexa adresiraju ovih N linija, dakle biće nam potrebno $index = \log_2 N = \log_2 2^{13} = 13$ bita indexa. Sada, za računanje bita offseta, bitno je razmotriti da li se adresira po riječi ili po bajtu unutar bloka. Obje opcije su validne, pa ćemo izračunati za obje.

Ako adresiramo po riječi, onda bite offseta računamo po formuli $offset = \log_2 word = \log_2 8 = 3$, pa nam treba 3 bita offseta. Za adresiranje po bajtu, trebalo bi nam $offset = \log_2 block = \log_2 64 = 6$ bita.

Napokon, bite taga računamo po formuli $tag = adr - index - offset$. Za adresiranje po riječi je $tag = 30 - 13 - 3 = 14$, a po bajtu je $tag = 30 - 13 - 6 = 11$ bita.

Za adresiranje keša po riječi nam je potrebno TAG=11b, INDEX=13b, OFFSET=3b, a po bajtu je potrebno TAG=14b, INDEX=13b, OFFSET=6b.

Zadatak 4: Mikroinstrukcije oglednog procesora

Za sljedeće mikroinstrukcije oglednog procesora napisati da li su validne. Ako nisu napisati zašto nisu, a ako jesu napisati vrijednosti svih kontrolnih signala mikroinstrukcije.

a) $f := lshift(sp + ir); mar = sp; rd$

b) $mar := lshift(mbr + a); if z then goto A1$

a) Ogledni procesor ima dvije sabirnice za čitanje iz registara, i to je dovoljno za dva registra **sp** i **ir** koji se čitaju. Bitno je naglasiti da nije svejedno koji će na koju sabirnicu, jer se **sp** istovremeno upisuje u **mar**, što znači da **sp** mora biti na sabirnici B, a **ir** na sabirnici A. Zbir **sp** i **ir** će ući u ALU, i shiftat se za jedno mjesto i ući u sabirnicu C, koja upisuje na adresu registra **f**.

A	CC	ALU	SS	MBR	MAR	RD	WR	ENC	REGC	REGB	REGA	ADRESA
0	00	00	10	0	1	1	0	1	1111	0010	0011	-----

Objašnjenje:

A: **mbr** ne igra ulogu u ALU,

CC: nemamo skokova na kraju instrukcije,

ALU: sabiramo, pa je 00,

SS: shiftamo ulijevo,

MBR: nemamo upis u **mbr** iz C sabirnice, tj. shiftera,

MAR: imamo upis u **mar**,

RD: čitamo vrijednost iz memorije i upisujemo u **mbr**,

WR: ne upisujemo vrijednost iz **mbr** nigdje,

ENC: upisujemo u memoriju vrijednost sa C sabirnice (ALUa),

REGC: adresa odredišta za upisivanje, sadrži adresu registra **f**

REGB: adresa izvorišta na sabirnicu B, sadrži adresu **sp**,

REGA: adresa izvorišta na sabirnicu A, sadrži adresu **ir**.

ADRESA: nebitno, jer nemamo skokova

b) Odmah vidimo da je problem što se izlaz iz ALU upisuje na **mar** koji uopće nema poveznicu na sabirnicu C.

Zadatak 5: MULTD instrukcija u oglednom procesoru

Koristeći mikroarhitekturu oglednog procesora, napisati, u minimalnom broju mikroinstrukcija, mikrokod faze izvršenja **MULTD** instrukcije koja će pročitati podatak sa memorijske lokacije specificirane sa donjih 12 bita instrukcije, pomnožiti pročitano vrijednost sa vrijednošću AC registra, i spremi rezultat u AC registar. **MULTD** će zamijeniti trenutnu **JNEG** (opkod 1100) instrukciju oglednog procesora. Modifikovati mikrokod oglednog procesora u Logisimu.

Ideja je ponavljati dodavanje vrijednosti akumulatora na sebe onoliko puta koliko iznosi vrijednost na adresi. Ponavljanje simuliramo smanjenjem iznosa na adresi za 1 (sabiranjem sa -1) dok ne postane nula. Treba napomenuti da ovaj proces radi jedno viška sabiranje, što znači da moramo smanjiti vrijednost iznosa na adresi za jedan prije početka ponavljanja. Ali, u tom slučaju ako je vrijednost na adresi jednaka nuli, otići će ta vrijednost u negativno, i u ovakvoj izvedbi, izazvati beskonačnu petlju. Tako da ćemo provjeriti vrijednost iznosa na adresi prije početka algoritma, i ako je nula, postaviti akumulator na nula i završiti množenje. Trenutni mikrokod, sa dodatnom modifikacijom, izgleda ovako:

```
00 00c00000 00506000 b013001c 24143313 3414040b 30000409 00c03000 00400000 f0110000 11a03100 60200000 3000040f 00c03000 00400000 e0111000 00c03000
10 00516100 981a0000 6011a100 34140419 30000417 30000100 68108300 50000116 60000000 3000041b 68108300 68118300 24143328 34140423 30000421 001a2300
20 60c0a007 001a2300 71a0a10a 30000426 001a2300 60c0a00d 001a2300 60c0a010 3414042e 3000042c 60000050 ffffffff 50000100 68108300 34140432 00127200
30 11a02000 68308300 34140441 3414043b 30000438 00c01000 00527200 60a0200a 00d22600 00400000 60a0100a 3000043e 00127200 71a0210a 00d22600 00400000
40 f0110000 34140449 30000446 00d22600 00400000 f0100000 101a0100 10110200 70120a00 3000044c 081a9300 6012a200 081a9300 181a0a00 601a6a4b 00000000
50 101a0100 00c03000 00400000 d01b0058 001bb700 00111a00 401bb700 60000055 70110500 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Zaokružena mikroinstrukcija u prethodnom bloku predstavlja mjesto gdje je počinjala instrukcija **JNEG**, i sada je zamijenjena skokom na adresu 0x50, odakle počinje **MULTD**. Razlog ovome je što, ako bismo trpali mikroinstrukcije množenja u ovaj mali prostor, morali bismo shiftati ostatak mikrokoda za nekoliko mjesta udesno, što predstavlja mukotrpan proces. Zbog toga skaćemo na adresu 0x50 (dec. 80) jer tu imamo prostora.

Mikrokod prati sljedeće mikroinstrukcije počevši od adrese decimalno 80 (hex. 0x50):

80. $a := ac$;

- $b = m(x)$; što je zapravo apstrakcija za naredbe

81. $mar := ir; rd$;

82. rd ;

83. $b := mbr$; *if* z goto 88;

84. $b := b + (-1)$;

85. $ac := ac + a$;

86. $b = b + (-1)$; *if* z goto 0;

87. goto 85;

88. *ac* := 0; *goto* 0;

Kodovi pojedinačnih mikroinstrukcija se mogu vidjeti u na prethodnoj slici počevši od adrese 80 (hex. 0x50). Ovaj mikrokod uspješno množi pozitivne (jer izgleda da ogledni procesor ne podržava negativne brojeve u memoriji) brojeve, kao i nulu, bila ona u akumulatoru ili na traženoj adresi.

Također je u Logisimu u projektu za ovaj zadatak napisan mali testni program koji množi 5 i 10 i smješta u akumulator:

LOC0 005 (7005);

ST0D 200 (1200);

LOC0 00a (700a);

MULTD 200 (c200);

Ovaj program se nalazi na adresi hex. 0x020, nakon skoka sa adrese 0x000.