VEGOVA | ELEKTROTEHNIŠKO-RAČUNALNIŠKA STROKOVNA ŠOLA IN GIMNAZIJA LJUBLJANA

Digitalna vezja v Minecraftu: od logičnih vrat do CPE

Raziskovalna naloga

Elektrotehnika

Avtor: Niko Maren, R 1. C

Mentor: Aleš Volčini

Ljubljana, 2023

Kazalo vsebine

Kazalo slik

Povzetek

V raziskovalni nalogi sem primerjal delovanje logičnih vrat in vezij v videoigri Minecraft z logičnimi vrati in vezji, ki jih uporabljamo v digitalni elektroniki. Zanimalo me je, ali so problemi, ki jih srečujemo v Minecraftu kakorkoli povezani s problemi, na katere bi naleteli, če bi se podstopili realizirati elektronsko digitalno vezje. Na koncu sem v Minecraftu sestavil tudi centralno procesno enoto in preizkusil njeno delovanje. Vsa vezja sem naredil iz osnovnih logičnih vrat ALI (angl. OR) in NE (angl. NOT) oz. njihovih kombinacij in s pomočjo Minecraftovih elementov, ki se obnašajo zelo podobno kot analogna vezja (ojačevalniki, komparatorji...).

Ključne besede: Minecraft, digitalna elektronika, logična vezja, centralna procesna enota

Abstract

In the research paper, I compared the operation of logic gates and circuits in the video game Minecraft with those used in digital electronics. I was interested in whether the problems encountered in Minecraft are in any way related to the problems we would face if we were to undertake the realization of an electronic digital circuit. In the end, I also assembled a central processing unit in Minecraft and tested its operation. All circuits were made from basic logic gates OR and NOT or their combinations, and with the help of Minecraft elements that behave very similarly to analog circuits (amplifiers, comparators, etc.).

Keywords: Minecraft, digital electronics, logic circuits, central processing unit

Uvod

Igro Minecraft sem začel igrati že pri šestih letih. Najprej me je navduševala kot raziskovalna igra (adventure game), ampak me je hitro začel zanimati mineral redstone (z njim lahko simuliramo logična vrata, vezja), saj sem na youtubu videl kaj vse se da narediti z njim. Videl sem, da so v Minecraftu začeli delati kalkulatorje, računalnike in prišli so celo do toliko sposobnih računalnikov, da se je lahko v Minecraftu igralo Minecraft (čeprav zelo okrnjeno različico). To me je zelo navdušilo in sem začel tudi sam ustvarjati manjša vezja. Kmalu sem ugotovil da se v realnosti srečujemo s podobnimi problemi kot pri ustvarjanju vezij v Minecraftu.

Hipoteze

Za delovne hipoteze sem si izbral naslednje trditve:

1. V realnosti imamo probleme s prešibkimi signali v vezjih. S podobnim učinkom se srečamo tudi v Minecraftu, saj lahko signal po redstonu potuje samo 15 kock. Tako moramo v obeh primerih iskati načine, kako se izogniti slabljenju signala.
2. V digitalni elektroniki poskušamo vezja čim bolje optimizirati, da so hitrejša, manjša, da porabijo za enako operacijo čim manj tranzistorjev in so tako tudi bolj energetsko učinkovita. Vezja želimo optimizirati tudi v Minecraftu, saj redstone deluje zelo počasi, in vsaka optimizacija veliko pripomore. Načini, s katerimi optimiziramo vezja v digitalni elektroniki, so uporabni tudi v Minecraftu in obratno.
3. V digitalni elektroniki poskušamo narediti vezja čim manjša, zato da porabimo čim manj materiala in prostora. Tudi v Minecraftu imamo razloge, da poskušamo narediti vezja čim manjša. V digitalni elektroniki to dosežemo z razporeditvijo komponent na tiskanih vezjih.
4. Pri optimizaciji vezij se v obeh primerih poslužujemo vseh mogočih trikov, še najpogosteje matematičnih, včasih pa tudi fizikalnih.
5. Tako kot v digitalni elektroniki različne komponente upočasnjujejo signal, tudi v Minecraftu različne komponente povzročajo zamude v signalih.
6. Tako kot imamo v digitalni elektroniki težave s presluhi med žicami, naletimo na podobne probleme tudi v Minecraftu.
7. Povsod se lahko pri ustvarjanju vezij lahko zmotimo, zato imajo lahko tako kot digitalna kot vezja v Minecraftu hrošče.
8. V digitalni elektroniki poskušamo vezja zasnovati na način, da jih kasneje lažje prilagajamo okolici. Tako lahko enako komponento oz. njen klon uporabimo v več vezij, kar zelo pospeši izdelavo večjih vezij. To velja tudi za Minecraft.
9. Če bom potrdil večino hipotez, je Minecraft odlična videoigra, kjer se lahko naučimo kako poteka izdelava vezij in kako delujejo tudi v realnem življenju.

Osnove delovanja računalnika

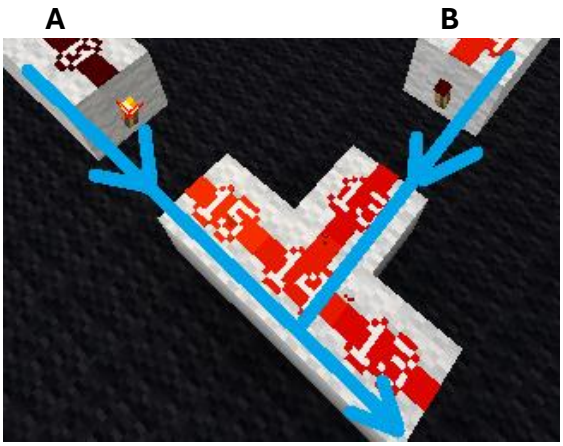
Računalnik lahko primerjamo z glasbeno skrinjico, le da namesto igranja po notah izvaja ukaze. Pri glasbeni skrinjici je glasba zapisana v npr. bobnu, medtem kot pri računalniku v npr. ROM-u. Procesor računalnika lahko primerjamo z delovanjem tistega dela glasbene skrinjice, ki proži inštrumente. Za razliko od glasbene skrinjice računalnik predvaja program, po katerem proži različna logična vezja v pravem zaporedju.

Logična vrata

Logična vrata (angl. *logic gates*) so osnovni gradnik digitalnih vezij. Izvajajo logične operacije nad biti. So temelj računalniške arhitekture, saj se iz več njih izdeluje ostala vezja. Najpomembnejša vrata so ALI (angl. *OR*), IN (angl. *AND*) in NE (angl. *NOT*), saj se da s pomočjo teh logičnih vrat sestaviti vsa ostala (temu pravimo da tvorijo t.i. polni nabor operatorjev). Podobno kot v digitalni elektroniki tudi v Minecraftu lahko naredimo logična vrata na več načinov.

Logična vrata ALI

V Minecraftu so najpreprostejša vrata ALI, saj lahko samo povežemo več redstona v sled. Taka sled se uporablja kot žica, je med seboj povezan prah redstona in prenaša signal. Vendar se signal širi v obe smeri, zato je potrebno uporabiti par trikov, da se to prepreči, saj lahko v nasprotnem primeru potuje signal nazaj do kakšnih drugih vrat, kar nočemo, saj potem tista vrata dobijo signal, čeprav tega nismo hoteli.



Slika 1: logična vrata ALI z blokiranjem signala s pomočjo vrat NE

Pravilnostna tabela:

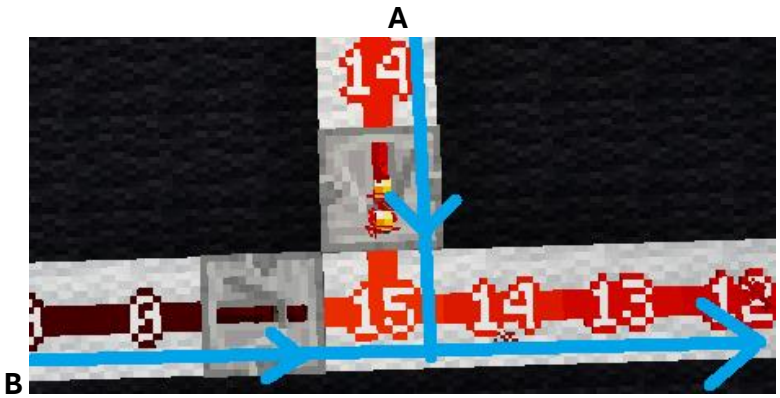
A	B	A ALI B
0	0	0
0	1	1
1	0	1
1	1	1

Prenašanje signala v neželeno smer lahko v Minecraftu preprečimo na več načinov:

- V nekaterih primerih nam tega sploh ni potrebno, saj tudi če gre signal nazaj po žici, si ustavi pri vratih, ki signala ne prepuščajo nazaj.

Na zgornji sliki modre puščice prikazujejo smer potovanja signala. Številke na redstonu (rdeča sled) prikazujejo moč signala. Vidimo, kako vrata NE (za njih se uporablja redstone bakla) ne prepuščajo signala nazaj.

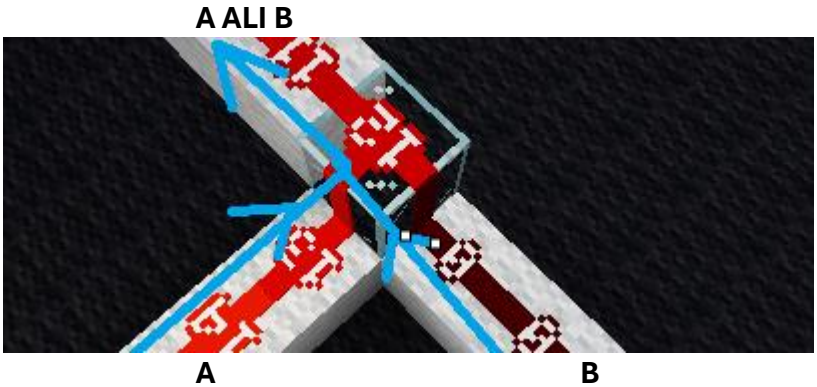
- Uporabimo ojačevalnike (angl. redstone repeater) ali podobne komponente, saj tako kot logična vrata tudi one prepuščajo signal samo v eno smer. Dobra stran pri uporabi ojačevalnikov je, da signal spet dobi svojo polno moč. Ta se vidi tako, da redstone sveti z različnimi jakostmi (največja je 15, z vsakim metrom, ki ga signal prepotuje pa se mu jakost zmanjša za 1). Slaba stran: vsi ojačevalniki in ostala logična vrata potrebujejo nekaj časa, da signal prepustijo čez, zato se temu načinu poskušamo čimbolj izogibati, da so vrata čim hitrejša.



Slika 2: logična vrata ALI z blokiranjem signala s pomočjo ojačevalnikov

Na zgornji sliki modre puščice prikazujejo smer potovanja signala, kjer vidimo, kako so uporabljeni ojačevalniki za preprečevanje prehajanja signala nazaj.

- Zadnji način je z uporabo mehkih blokov, to so kocke, ki ne prevajajo redstone signala, največkrat se uporablja steklo in polovične kocke (angl. slab)), saj ko iz kocke nižje napeljemo redstone na 1 kocko višje, dobimo efekt, da signal potuje samo v smeri navzgor. Ta način je veliko bolj zaželen od prejšnjega, saj ne povzroča zakasnitev signala.



Slika 3: logična vrata ALI z blokiranjem signala s pomočjo mehkega bloka

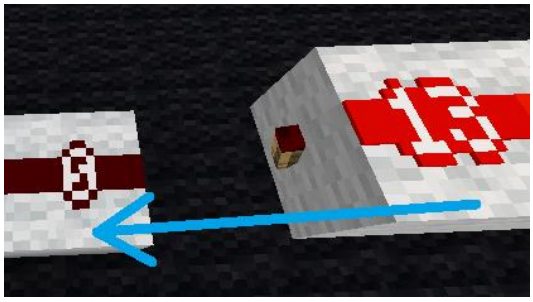
Tako kot prej tudi tu modre puščice označujejo smer signala. Prikazan je primer uporabe stekla v vlogi mehkega bloka, ki preprečuje povraten signal.

Logična vrata NE

Druga najpogostejša vrata so vrata NE, saj za njih preprosto uporabimo samo redstone bakle (angl. *redstone torch*). Ko blok, na katerega je postavljena taka bakla ni pod vplivom signala, je bakla prižgana (slika 4), v nasprotnem primeru ugasnjena (slika 5).

Blok, ki sledi bakli, prejema od nje signal in ga posreduje dalje.

Slika 4: logična vrata NE s prižganim izhodom



Slika 5: logična vrata NE z ugasnjenim izhodom

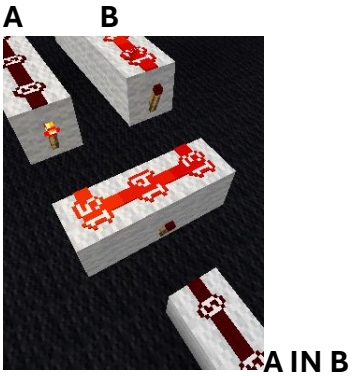
Modre puščice označujejo smer potovanja signala. Prikazana so vrata NE v obeh stanjih.

Pravilnostna tabela:

A	NE A
0	1
1	0

Logična vrata IN

Vrata IN so vrata, ki so sestavljena iz več drugih vrat po De Morganovem zakonu. Sestavljena so iz enih vrat ALI in treh vrat NE. Najprej oba signala negiramo, nato ju speljemo v vrata ALI in potem še enkrat negiramo.



Slika 6: logična vrata IN z enim prižganim signalom na vhodu

Slika 7: logična vrata IN z dvema prižganima signaloma na vhodu

Slika 8: logična vrata IN z ugasnjenima signaloma na vhodu

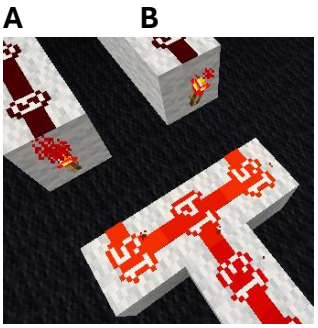
Prikazana so vrata IN v treh stanjih vhodov.

Pravilnostna tabela:

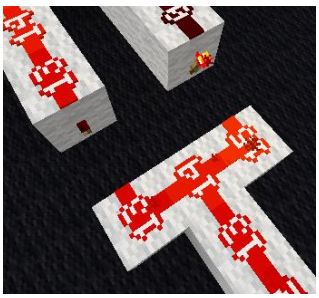
A	B	A IN B
0	0	0
0	1	0
1	0	0
1	1	1

Logična vrata NEIN

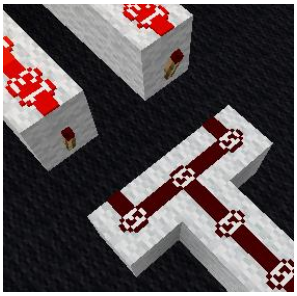
Vrata NEIN (angl. NAND) so vrata IN, samo da jih na koncu ne negiramo. To je tudi edino, kar jih razlikuje od vrat IN, opisanih v prejšnjem poglavju. Tako prihranimo eno redstone baklo.



Slika 9: logična vrata NEIN z ugasnjenima siignaloma na vhodu



Slika 10: logična vrata NEIN z enim prižganim signalom na vhodu



Slika 11: logična vrata NEIN z dvema prižganima signaloma na vhodu

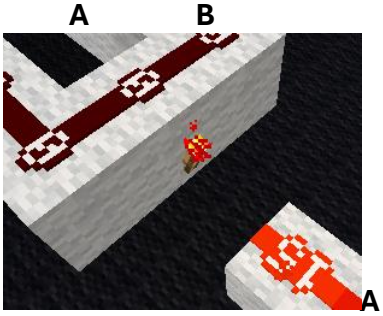
Prikazana so vrata NEIN v treh stanjih vhodov.

Pravilnostna tabela:

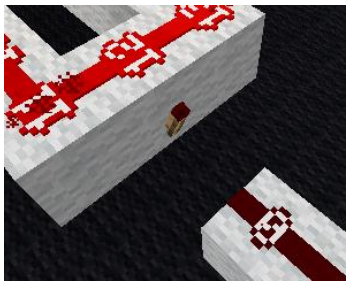
A	B	A NEIN B
0	0	1
0	1	1
1	0	1
1	1	0

Logična vrata NALI

Vrata NALI (angl. *NOR*) so vrata ALI, samo z negiranim izhodom. Ker so vrata ALI najpreprostejša vrata v Minecraftu, so tudi vrata NALI zelo preprosta.



Slika 12: logična vrata NALI z aktiviranim izhodom



Slika 13: logična vrata NALI z neaktiviranim izhodom

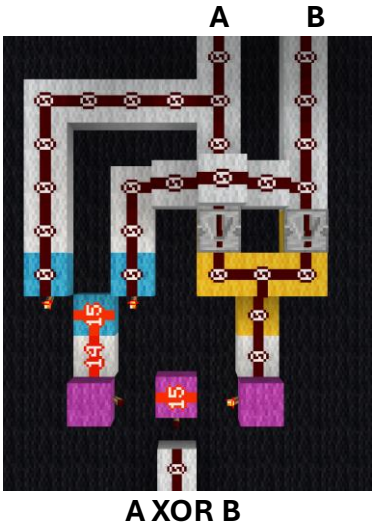
Prikazana so vrata NALI v dveh stanjih vhodnih bitov.

Pravilnostna tabela:

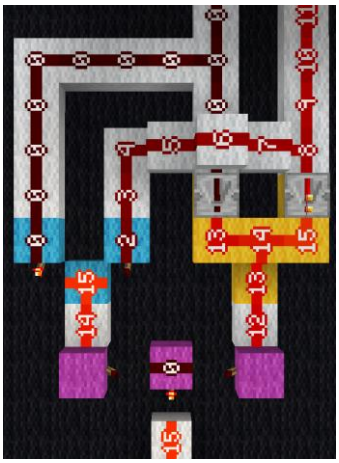
A	B	A NALI B
0	0	1
0	1	0
1	0	0
1	1	0

Logična vrata XOR

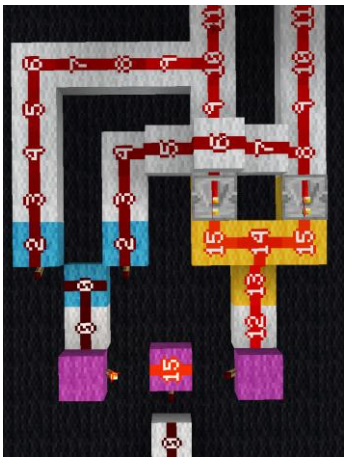
Vrata za ekskluzivni ALI oz. vrata XOR so najkompleksnejša vrata, saj so sestavljena iz vrat enih vrat NEIN, enih vrat ALI in enih vrat IN. Najprej se oba vhodna signala razdelita v na dve strani, na eni strani gresta signala v vrata NEIN, na drugi pa v ALI. Na koncu gresta izhodna signala iz obeh prejšnjih vrat v vrata IN.



Slika 14: logična vrata XOR z ugasnjenima signaloma na vhodu



Slika 15: logična vrata XOR z enim prižganim signalom na vhodu



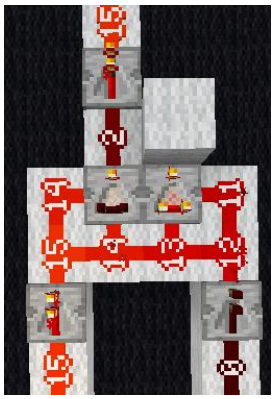
Slika 16: logična vrata XOR z dvema prižganima signaloma na vhodu

Na zgornjih slikah so prikazana vrata XOR v treh stanjih. Z belimi kockami so označene povezave med vrati, z modrimi vrata NALI, z rumenimi vrata ALI, z rožnato pa vrata IN. Uporabljena je vrst vrat ALI z ojačevalniki, saj mora na vrata IN priti signal z ob enakem času, saj lahko v nasprotnem primeru dobimo napačno vrednost na izhodu za kratek čas. S to težavo se soočamo tudi v realnem življenju. Prikazan model XOR vrat je zelo velik in počasen.

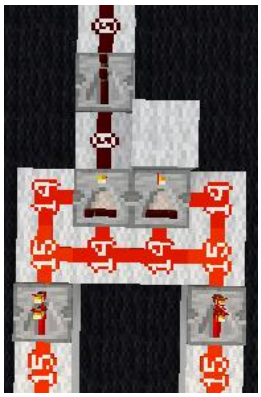
V Minecraftu poznamo še drug model XOR vrat, ki je manjši in ponavadi hitrejši. Deluje s pomočjo komparatorjev.

Komparator v Minecraftu je komponenta, ki primerja dva signala. V primerjalnem načinu deluje tako, da v primeru, če je signal B šibkejši ali enak signalu A, oddaja izhodni signal z močjo signala A, sicer pa ga blokira. V odštevalnem načinu oddaja signal z močjo signala A – B, a le če je signal A močnejši od signala B (in je razlika tako pozitivna). V nasprotnem primeru signal ostane na ničli.

Slika 17: logična vrata XOR s komparatorji in ugasnjenima signaloma na vhodu



Slika 18: logična vrata XOR s komparatorji in enim prižganim signalom na vhodu



Slika 19: logična vrata XOR s komparatorji in dvema prižganima signaloma va vhodu

Na slikah je prikazan model XOR vrat s komparatorji v odštevalnem načinu in v treh različnih stanjih vhodnih signalov.

Komparator sprejema signal A na svoji spodnji strani, ob straneh sprejema signal B, oddaja pa ga na zgornji strani (gledano glede na postavitev na slikah zgoraj, kjer na sredinah vidimo po dva komparatorja). Signal pride do vrat tako, da ima enako moč pri komparatorju, ki mu je bližje. S tem dosežemo da komparator, ki je bližje ne oddaja signala, saj je razlika med jakostjo signala A in signala B enaka 0.

Na komparatorju, ki je bolj oddaljen od vhoda, pa vidimo da je signal na vhodu B za 2 stopnji šibkejši, razlika A-B je 2, in zato oddaja signal z močjo 2. Kot posledica tega rabimo v nekaterih primerih takoj za vrati dodati ojačevalnik.

Če pa na vhoda A in B dobimo dovolj močna in po jakosti ne preveč različna signala, potem sta pri obeh komparatorjih signala na vhodih enaka, in noben izmed njiju ne oddaja signala. Da to res deluje in da dosežemo, da noben izmed komparatorjev ne oddaja signala, se smeta vhodna signala razlikovati za največ 2 stopnji jakosti. Zato moramo tik pred vhode vrat včasih postaviti ojačevalnike. Kocka, v katero komparator oddaja signal, nato prenese signal na redstone zraven nje.

Ta vrata so eden izmed primerov, ko se poskušamo poslužiti vseh možnih trikov, samo da bi bilo določeno vezje hitrejše, manjše in manj kompleksno. Zato v Minecraftu skoraj vedno uporabljamo model XOR vrat s komparatorji.

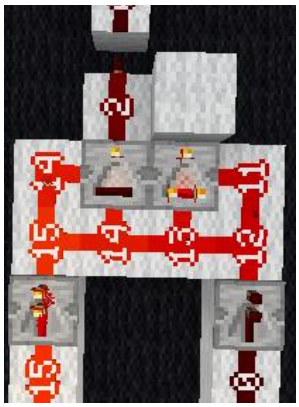
Pravilnostna tabela:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

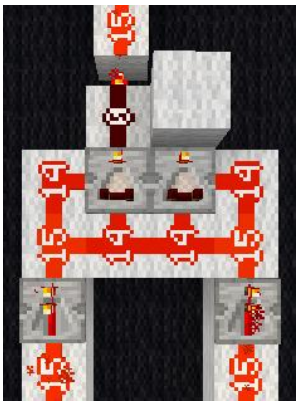
Logična vrata XNOR

Logična vrata za ekvivalenco ali vrata XNOR so vrata XOR, samo da signal na koncu negiramo. Praviloma za njih vedno uporabljamo model XOR vrat s komparatorji, opisanem v prejšnjem poglavju.

Slika 20: logična vrata XNOR z ugasnjenima signaloma na vhodu



Slika 21: logična vrata XNOR z enima prižganim signalom na vhodu



Slika 22: logična vrata XNOR z dvema prižganima signaloma na vhodu

Na slikah vidimo model XNOR vrat s komparatorji in pri treh različnih stanjih vhodov. Pri tem modelu vrat na izhodu ne potrebujemo ojačevalnika, saj v ta namen služijo že vrata NOT.

Pravilnostna tabela:

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Osnovna vezja

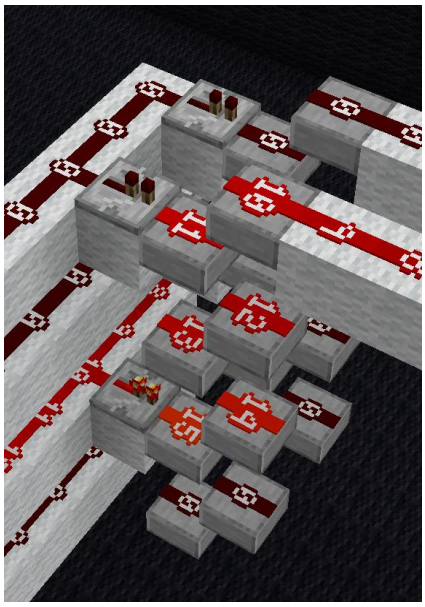
Vsa logična vezja v Minecraftu so sestavljena iz logičnih vrat in s pomočjo manjših trikov, ki so mogoči v Minecraftu.

Enkoder

Enkoder (angl. *encoder*) je logično vezje, ki pretvori poljubno število bitov širok vhod v poljubno število bitov širok izhod. Zahteva se, da je na vhodu prižgan natanko en bit, ki tako predstavlja natanko določeno kombinacijo izhodnih bitov. Določen vhodni bit zakodiramo tako, da povežemo linijo vhodnega bita z izhodnimi linijami tam, kjer hočemo da so zakodirane enice.

Slika 23: horizontalni enkoder

Na tej sliki je prikazan 4 na 2 enkoder. Narejen je v horizontalnem modelu. Iz vhodnih bitov povežemo signal na izhodne na način, da najprej vse vhodne bite negiramo, in nato na vseh bitih, ki hočemo da preide signal na izhodne bite, povežemo s pomočjo redstone bakel (ki hkrati bite tudi negirajo), kot je pokazano na sliki levo (v zgornjem levem kotu so vhodni biti, v spodnjem levem kotu izhodni). Trenutno je prikazano, kako se drugi vhodni bit zakodira v izhodne bite. Ta način zahteva več časa, ker gre signal čez dve NE vrati, vendar lahko na tak način vezje naredimo veliko manjše. Zato je poleg velikosti tudi časovno skoraj enako oz. pri večjih vezjih še bolj učinkovito, saj je manjše in ne potrebujemo veliko ojačevalnikov. Enkoder na sliki, kodira vhodno stanje štirih linij na dva bita. Tako štiri možna stanja na vhodu spravimo v binarno vrednost od 00 do 11.



Slika 24: vertikalni enkoder

Obstaja še vertikalen model enkoderja. Vhodni signali pridejo noter vertikalno en nad drugim, izhodni pa gredo ven še vedno horizontalno. Ta model je hitrejši, saj lahko povezujemo vhodne signale z izhodnimi s samo enim ojačevalnikom. Za prenos podatkov navpično gor se običajno uporablja stolpe iz polovičnih kock. Uporabimo tudi katerikoli druge »mehke« bloke, ker poleg tega da po mehkih blokih potuje signal samo navzgor, imajo ti bloki tudi to lastnost, da lahko redstone speljemo poševno pod njimi.

Na sliki levo je prikazan tak model enkoderja. Zakodiran je na isti način kot prejšnji, enako je tudi število vhodnih/izhodnih bitov, trenutno kaže kako se zakodira drugi vhodni bit. Na levi strani slike so vhodni podatki, na desni izhodni. Ta model se uporablja veliko več kot horizontalni, saj se signale v Minecraftu običajno napeljuje vertikalno enega nad drugim, predvsem zaradi tega, ker je tudi večina ostalih vezij narejena tako.

Dekoder

Slika 25: dekoder

Dekoder (angl. *decoder*) je vezje, ki dekodira več vhodnih bitov v izhodne tako, da je vedno prižgan natanko en izhodni bit.

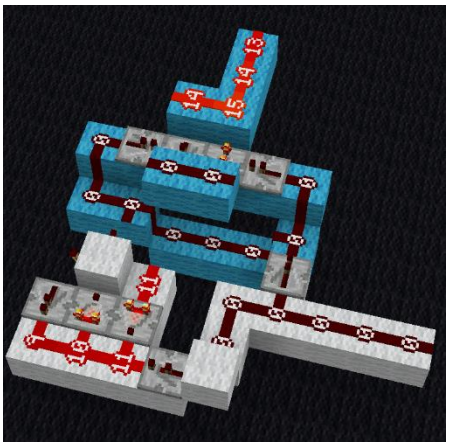
Na sliki levo je prikazan 2-na-4 dekoder. Vhodna bita sta na levi strani slike, izhodni na desni. Vsak stolpec predstavlja en izhodni bit. Na vhodu v posamezen stolpec postavimo NEIN vrata, ki se aktivirajo ob natanko eni izbrani kombinaciji vhodnih signalov. V ta namen redstone baklo postavimo na vsak bit, kateri hočemo da je prižgan ob aktivaciji izhodnega bita, ojačevalnik pred vse ostale. Na koncu se vse bite negira, da dobimo želeni bit prižgan ostale ugasnjene. Ker rabijo do vrat NE priti signali istočasno, saj bi se v nasprotnem primeru na izhodu za kratek čas aktiviral napačen bit, nam ojačevalnik pride prav, saj pri tem povzroča enako zakasnitev v signalu kot redstone bakla.

Ura

Urino vezje ali kratko ura (angl. *clock*) je vezje, ki daje dogodkom v procesorju takt. Časovnemu razmaku med ponovitvami signalov pravimo cikel.

V Minecraftu imajo vse redstone komponente vedno enako zakasnitev ne glede na vrsto dogodka. Zato lahko zelo dobro izračunamo potrebno dolžino cikla in temu primerno

prilagodimo urino vezje. Ko pripravljamo večja digitalna vezja kot je npr. centralna procesna enota, je mnogokrat ključnega pomena potreba, da signali do komponente prispejo v istem trenutku.



Slika 26: ura

Izbira frekvence urinega vezja je povezana z mehaniko same igre.

Najpočasnejša izmed redstone komponent je redstone bakla. Preklaplja lahko s frekvenco 5Hz, a samo za 13 ciklov. Naslednji cikel mora biti zaradi mehanike igre daljši. Tako je uporabna frekvenca v povprečju 4.667Hz. Brez tega podaljšanja bakla preide v nasičenje in se za dalj časa neha odzivati.

Frekvenco 5Hz sem dosegel z eno uro (modri del), a sem moral narediti še eno uro zraven (beli del), ki ustavlja prvo uro za en cikel vsakih 14 ciklov. Izhodni signal je na zgornji strani slike, na desni strani je signal, ki ustavi obe uri za čas, ko je prižgan.

Signal lahko negiramo tudi s komparatorji, ampak je tako vezje v primerjavi z baklo ogromno in pri velikem številu postane taka izvedba pretirano požrešna s prostorom.

Kompleksna vezja

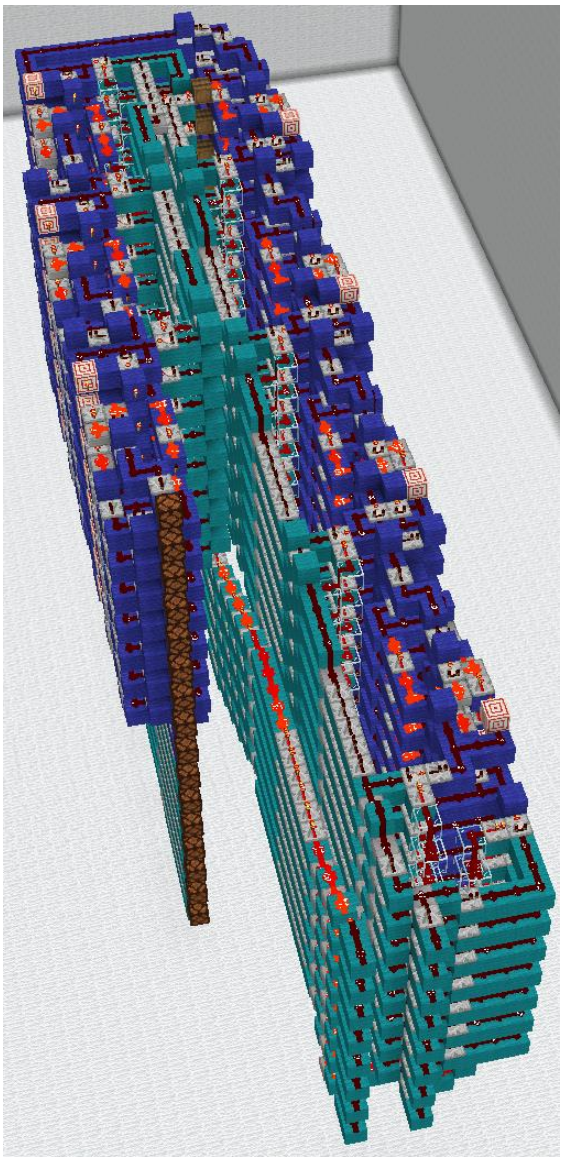
Večja digitalna vezja vsebujejo mnogo manjših enakih vezij, ki jih potem samo drugače povezujemo med seboj. Na tak način ustvarjamo večja vezja tudi v Minecraftu. V Minecraftu se signali največkrat prenašajo tako, da je en bit nad drugim, ker je tudi večina vezij najlažje narediti v pokončni izvedbi. Tako v pokončni izvedbi včasih naredimo tudi tista vezja, ki so sicer preprostejša v vodoravni izvedbi. S tem prihranimo ogromno prostora in naredimo vezje bolj kompaktno.

Aritmetična in logična enota

Aritmetična in logična enota (angl. Arithmetic and Logic Unit (ALU)), je vezje, ki v procesorju izvaja računske operacije. Najprej sem izdelal vezja za posamezne računske operacije ločeno, nato sem jih povezal med sabo.

Vezje za množenje

Pri vezju za množenje sem imel na izbiro dva načina. Lahko bi ga naredil manjšega tako da bi se biti iz izhoda seštevalnika prenašali na vhod. Lahko pa bi ga naredil večjega, kjer bi šli biti iz prejšnje operacije vedno v nov seštevalnik. S tem bi dosegel delovanje v obliki cevovoda (angl. *pipeline*). Drži, da je tako vezje veliko veliko večje, ampak se je v tem primeru to splačalo, saj bi vezje v obeh izvedbah potrebovalo 4.4 sec, ampak pri izvedbi s cevovodom lahko pošljem nove podatke že čez 0.2 sec.



Slika 27: vezje za množenje

Vezje zmnoži dve 8 bitni števili v 16 bitno. Vezje pošlje prvo številko v vsak seštevalnik, vsakič zamaknjeno za en bit v desno. Seštevalnik prišteje prvo število k rezultatu le, če je ustrezni bit druge številke enak 1. Od vsakega seštevalnika gre spodnji bit k končnemu rezultatu.

Pri tem vezju sem moral dodajati zelo veliko zakasnitev, saj če sem hotel doseči v cevovodu paralelizacijo, sem moral dodati toliko zakasnitve na ostalih bitih, da so vsi biti prišli ob enakem času do seštevalnikov. Uporabljal sem paralelne (vzporedne) seštevalnike, saj so hitrejši in vsi biti na izhod pridejo istočasno, to pa zelo pomaga pri implementaciji cevovoda.

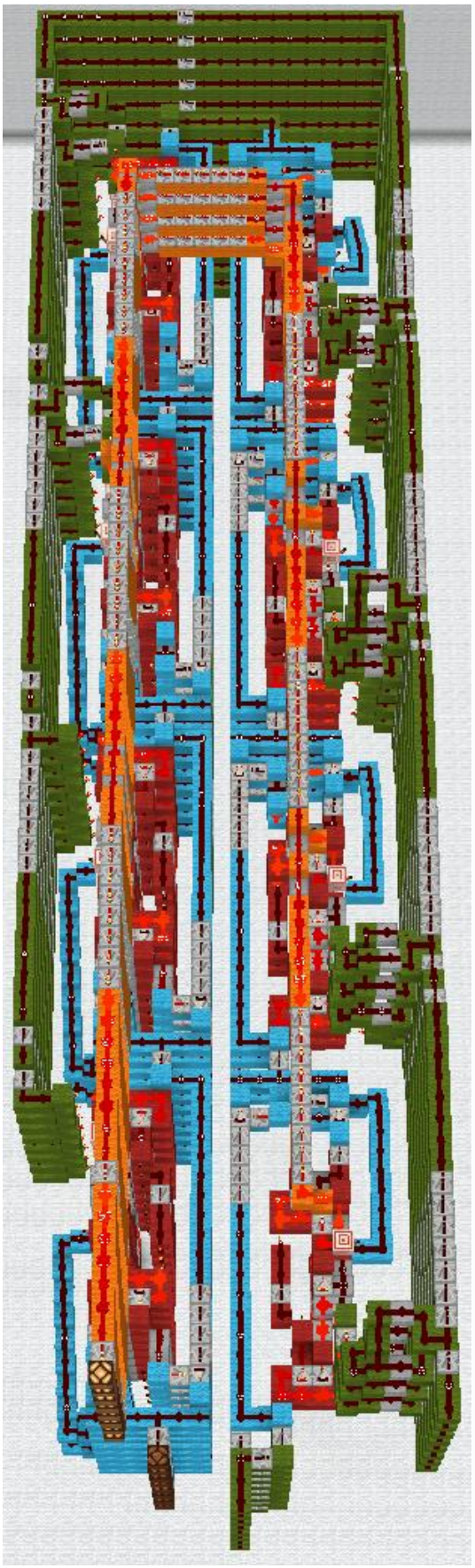
Seštevalnike sem postavljal tako, da so izhodni biti prišli čim bližje vhodnim, zaradi lažje povezave z ostalimi vezji. Na spodnji strani slike 27 so vhodni biti, izhodi pa malo višje na levi strani, označeni z redstone svetilkami (rjava kocka, ki zasveti če prejme redstone signal). S temno modro so označeni seštevalniki, s turkizno pa vse ostalo.

Vezje za deljenje

Pri gradnji vezja za deljenje sem imel podobne opcije kot pri vezju za množenje. Spet sem se odločil za večjo varianto, ker če sem že vezje za množenje naredil s cevovodom, je edino smiselno da na tak način naredim tudi vezje za deljenje.

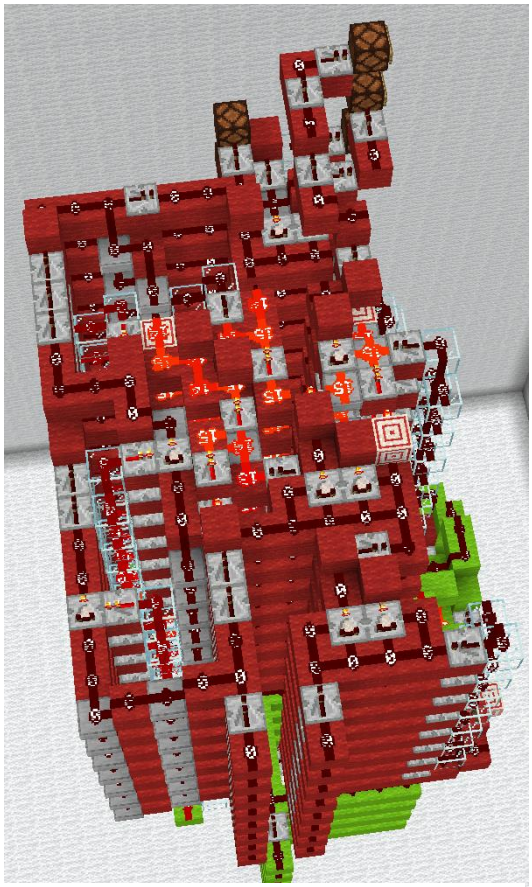
Pri ustvarjanju tega vezja sem imel podobne probleme kot pri ustvarjanju vezja za množenje, le da je tu glavna razlika ta, da sem namesto seštevalnikov uporabil pogojne odštevalnike.

Vezje deli dve 8-bitni števili v 8-biten količnik in 8-biten ostanek, saj če potrebujemo da nam zdeli na 8 bitov decimalk natančno, lahko še enkrat delimo ostanek z isto številko kot prvič. Na zelenem delu potuje delitelj, spodaj na sredini je deljenec. Rdeči so seštevalniki, modro je vezje, da seštevalnike spremeni v pogojne odštevalnike. Po oranžnem delu potuje količnik (skrajni levi stolp redstone svetilk na sliki 28 v levem spodnjem kotu). Ostanek je število, ki ga odda zadnji pogojni odštevalnik (stolp redstone svetilk, levo od sredine slike, desno od količnika).



Slika 28: vezje za deljenje

Vezje za seštevanje, odštevanje in bitne operacije

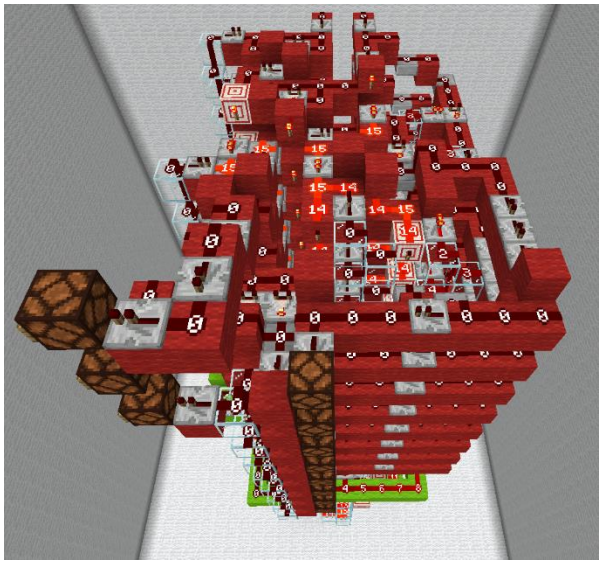


Slika 29: vhodna stran vezja za seštevanje, odštevanje in bitne operacije

Drži, da se bitne operacije izvajajo hitreje od seštevanja in odštevanja, a ker sem naredil vezje s cevovodom, sem tudi te operacije upočasnil na hitrost odštevanja, saj bi v nasprotnem primeru lahko prišlo do zmede na izhodu, ker bi na izhod prišli biti seštevanja, odštevanja in bitnih operacij istočasno. Najprej sem naredil vezje za seštevanje, a ker je zelo lahko narediti menjavanje med seštevanjem in odštevanjem, sem implementiral ti dve operaciji s samo enim seštevalnikom. Uporabljal sem paralelni seštevalnik (angl. *carry look-ahead adder*).

Za določene bitne operacije sem lahko samo malo spremenil odštevalnik, npr. za XOR sem samo dodal signal, ki blokira prenos pri seštevanju. Takih trikov se poslužujemo tudi v digitalni elektroniki pri izdelavi vezij, kar potrjuje mojo hipotezo.

Za večino ostalih bitnih operacij sem moral uporabiti dodatna vezja, tudi za zamik bitov (angl. bitshift) v desno, saj sem imel premalo prostora pri vhodu, da bi naredil vezje, ki prenese eno vhodno številko na oba vhoda.



Slika 30: izhodna stran vezja za seštevanje, odštevanje in bitne operacije

Najtežji del je bil povezati dekoder na signale vezja, saj sem določene kontrolne signale (npr. prenos, omogoči prenos, omogoči seštevanje itd.) moral povezati z več ukazi naenkrat. Tako je na primer bilo potrebno povezati kontrolni signal za omogočanje prenosa z vezjem za seštevanje, odštevanje itd. Signal sam pa je prišel iz dekoderja na podlagi strojne kode inštrukcije. Dekoder in vse povezave sem poskusil stlačiti na čim manjši prostor, da bi potreboval ukaz čim manj časa da aktivira kontrolne signale (za npr. omogočanje prenosa).

Tu sem imel največ problemov s povezavami, saj so bile sledi redstona zelo blizu druga drugi in je bilo težko ločiti te sledi med sabo. S tem problemom se

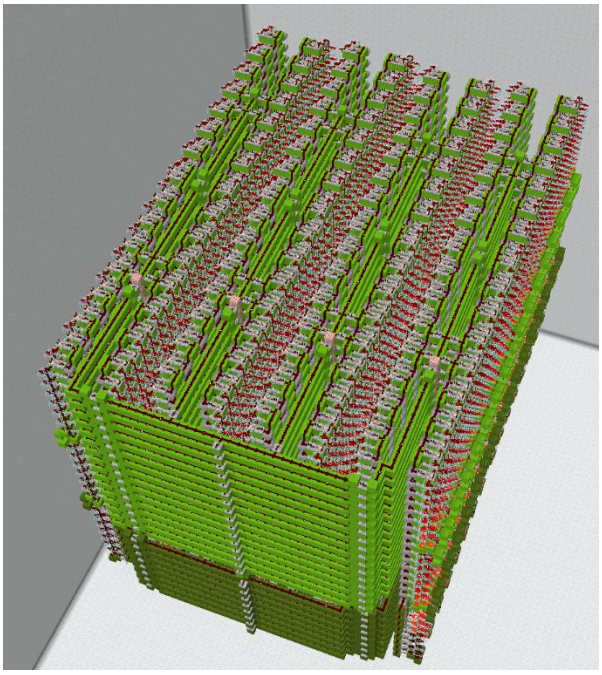
soočamo tudi v digitalnih vezjih. Tudi tam so včasih povezave preblizu ena drugi in prihaja do motenj in presluhov, kar potrjuje mojo hipotezo 5.

Imel sem tudi velike probleme z zakasnitvami signala oz. sinhronizacijo, saj morajo do različnih delov vezja priti signali istočasno ali s primerno zakasnitvijo. Npr. signal za omogočanje prenosa mora prispeti kasneje, kot signal za menjavo med odštevanjem in seštevanjem itd. Dodal sem tudi vezje za zastavice (angl. *flags*) in ukaze, ki jih aktivirajo ali deaktivirajo. Tako imam zastavice za prekoračitev (angl. *overflow*), ni negativno, ni nič.

Slika 29 prikazuje vhodno stran vezja, slika 30 pa izhodno. Rdeči del računa s števili, zeleni del je dekoder.

ROM

Pri ustvarjanju procesorja sem se odločil, da bom program shranjeval v ROM-u, saj si nisem hotel otežiti dela. Pri ustvarjanju ROM-a sem najprej naredil celico za 1 bit.



Slika 31: ROM

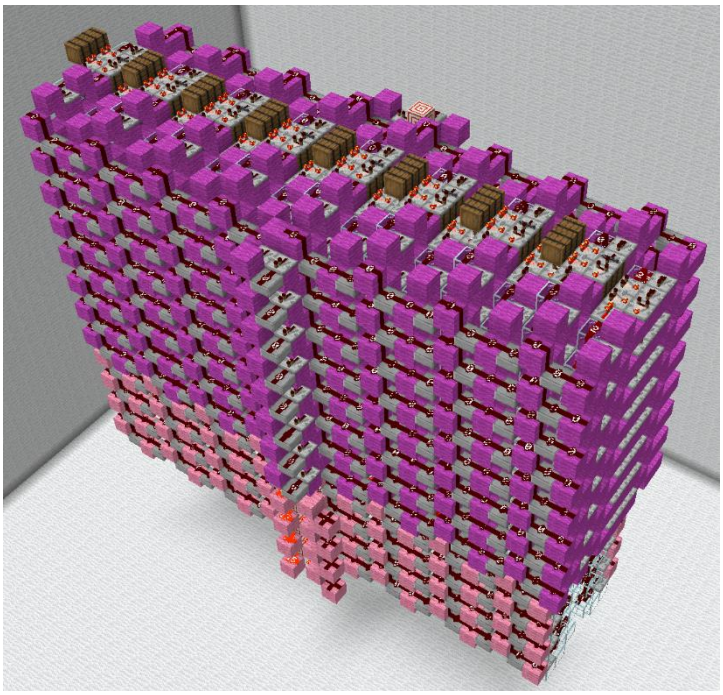
Zasnoval sem jo tako, da se jih lahko naloži v stolp in da se lahko potem tudi nastali stolpi stisnejo čim bolj skupaj. Nastalo celico sem samo kopiral, do željene velikosti ROM-a. Ta način je podoben tudi implementacijam v digitalni elektroniki, saj je s tem poenostavljena proizvodnja in manjša možnost za nastanek napak.

Na koncu sem dodal še dekoder, ki izbira stolp v katerem je ukaz. V ROM zapišemo program tako, da v sode shranimo neke predmete. Sodi so v Minecraftu posode za shranjevanje. Zraven damo komparator. Če niso prazni, to komparator zazna in odda signal z jakostjo večjo od 0. V sode damo največ toliko predmetov, da je na izhodu komparatorja jakost 1. Ta signal pa nato primerjamo signalom za izbiro stolpa (ki z oddaljenostjo slabi). Tako ne potrebujemo dodatnih redstone ojačevalnikov, da bi komparatorju preprečili oddajanje signala medtem, ko stolpec ni izbran.

Svetlozeleni del vezja na sliki 31, je del, kamor se shranjujejo podatki, temnozeleni del pa dekoder. Vanj lahko shranimo 256 16-bitnih ukazov z operandi.

Registri

Registre (angl. *register file*) sem naredil po zelo podobnem postopku kot ROM, saj sem rabil celicam spremeniti samo način shranjevanja podatka, in sicer tako, da sem dodal tudi funkcijo da se podatek lahko tudi shrani.



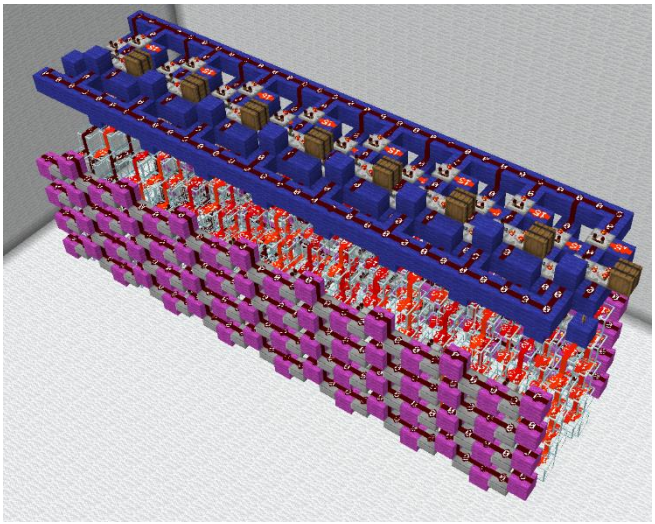
Slika 32: registri

Registrov nisem želel narediti v tako veliki količini kot ROM, saj bi to podaljšalo dostopni čas. Celice sem naložil samo eno na drugo in potem te stolpce enega zraven drugega. Celico sem zato zasnoval tako da je čim ožja, čim nižja in da ma čim krajše dostopne čase. Na sliki 32 je vijolični del vezja del, kamor se shranjujejo podatki, rožnati del pa dekoder. V ta register lahko shranimo 16 (oziroma 15, saj dekoder vedno oddaja signal za pisanje eni celici) 8-bitnih podatkov.

Register za zastavice

Tudi register za zastavice sem naredil po zelo podobnem postopku kot register, samo da ker so zastavice enobitne, sem rabil celico izdelati samo čim ožjo, saj jih nebom naložil več v višino. S tem sem lahko pridobil tudi nekaj malega na hitrosti. Modri del na sliki 33 je del, kamor se shranjujejo zastavice, vijolični del pa je dekoder.

V ta register lahko shranimo 16 zastavic, oz. 15 saj se v eno celico vedno piše zaradi načina dekodiranja naslova.



Slika 33: register za zastavice

Centralna procesna enota

Najprej sem se moral odločiti, koliko biten procesor hočem narediti. Odločil sem se za 8-bitnega, saj je pri večini vezij večja zakasnitev v signalu, če delamo z več kot osmimi biti na enkrat (problemi zaradi slabljenja redstone signala).

Izdelavo centralne procesne enote (angl. CPU) sem začel tako, da sem najprej ustvaril vsa vezja, ki jih taka enota potrebuje (register, ALE...). Ne glede na to, ali bi se odločal med princetonsko ali hardvarsko arhitekturo, so ta vezja znotraj procesorja enaka.

Pri izdelavi vseh vezij sem pazil, da podpirajo frekvenco procesorja do 4.667 Hz. To je tudi teoretična najvišja frekvenca, ki je primerna za CPE glede na omejitve igralnega pogona igre Minecraft. To hitrost izvajanja ukazov sem dosegel z uporabo cevovodov, saj če naslednji ukaz ne potrebuje rezultatov od prejšnjega, ne rabimo čakati da se izvede do konca.

Pri vezjih nisem imel večjih problemov z doseganjem te hitrosti, saj so cevovodi kar dobro realizirani. S tem sem imel problem edino pri števcu programa (angl. *program counter*), saj tu cevovod ne pride v poštev, ker mu rabimo prišteti 1 pri vsakem izvedenem ukazu. Ker števec programa vedno prišteje samo 1 (seveda ima tudi opcijo da nastavimo številko na nekaj drugega, a to zelo redkeje kot prištejemo 1), hitrost te operacije ni bila problem. Sem pa moral uporabiti malce drugačno vezje kot seštevalnik, ki je malce hitrejše, njegovo izvedbo sem si prikrojil po izvedbi uporabnika mattbatwings opisanega v youtube posnetku[3]. Tako sem uspel števec programa spraviti na hitrost 2.5 Hz, a je bilo še vedno prepočasi. Težavo sem rešil tako, da števec programa določa samo zgornjih 7 bitov številke, in da se aktivira na vsak drug signal ure, najmanjši bit sem pa povezal direktno z uro. Sicer nam to omogoča samo poljubno spreminjanje zgornjih sedmih od osmih bitov, a se mi zdi vseeno ugodna zamenjava za pospešitev izvajanja ukazov.

Odločil sem se za varianto harvardske arhitekture, saj bo program v ROM-u, operandi pa v samih instrukcijah in registrih (programski in podatkovni pomnilnik sta torej ločena).

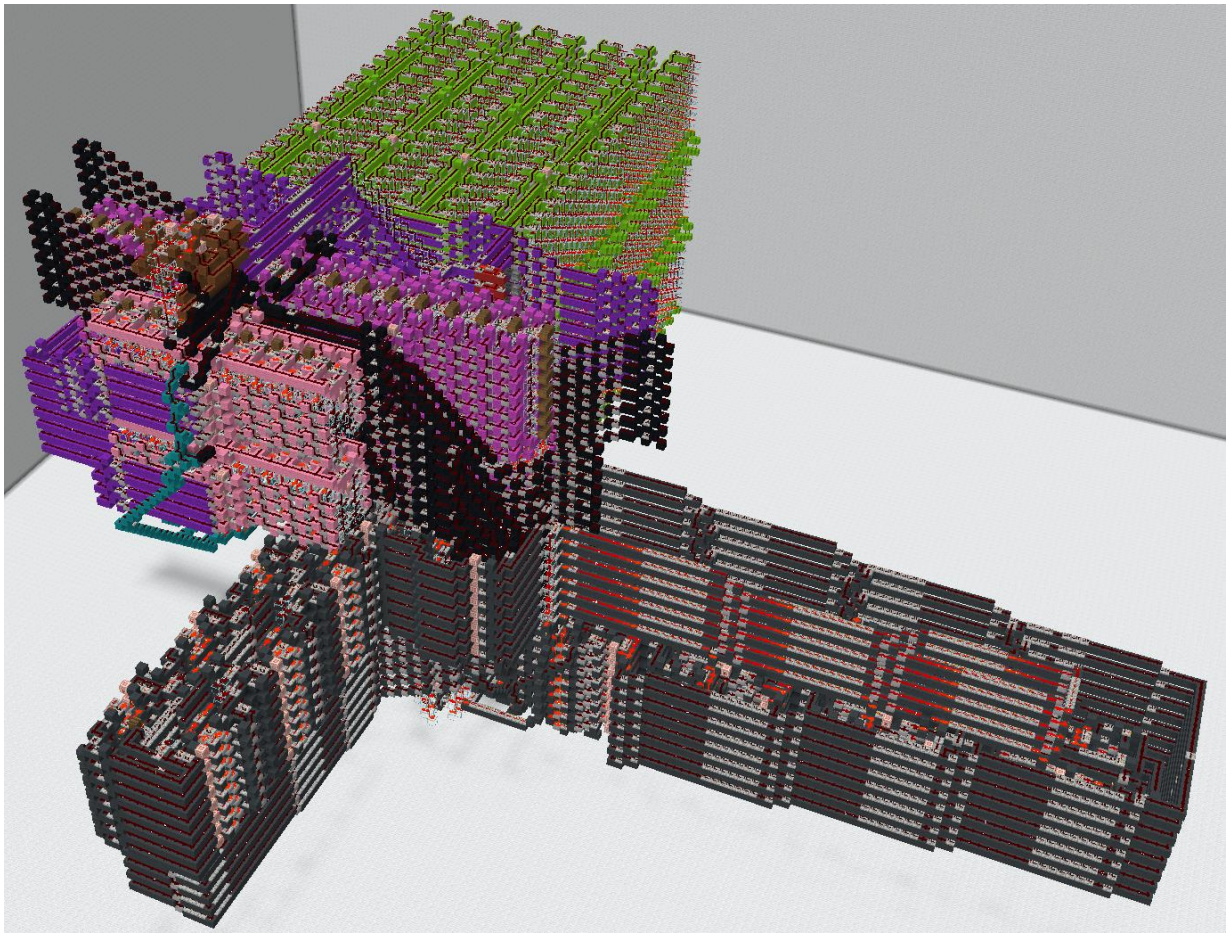
Ko sem začel povezovati vezja med sabo, sem takoj naletel na težavo. ALE deluje s frekvenco 4.677 Hz in bere dva podatka istočasno, a ker registri delujejo z enako hitrostjo, za ALE pa je to za polovico prepočasi. To sem rešil tako, da sem naredil dva bloka registrov. ALE je prebral iz vsakega bloka (recimo jima blok A in blok B) prebral po eno številko, rezultat pa shranil nazaj v blok A. S tem sem moral dodati tudi operacijo, ki prekopira register iz bloka A v blok B (obratno ni potrebno, saj lahko registru B samo prištejemo 0, saj se rezultat nato shrani v register A).

ALE shrani zastavice v poseben register. Tudi tu sem uporabil 2 nabora registrov za zastavice iz podobnih razlogov kot pri ALE. A tu je bila še ena težava. Ker ALE z vsako operacijo shrani 3 zastavice, sta 2 registra prepočasna za shranjevanje. To sem rešil tako, da sem signal tretje zastavice zakasnil za en cikel ure. Sicer drži, da če uporabimo 2x zaporedoma ukaz, ki aktivira ALE, da se zadnja zastavica povozi, a to ni problem, saj zastavice potrebujem samo pri instrukciji JumpIf (preskoči na drugo vrstico programa, če je izbran bit v registru za zastavice enak 0).

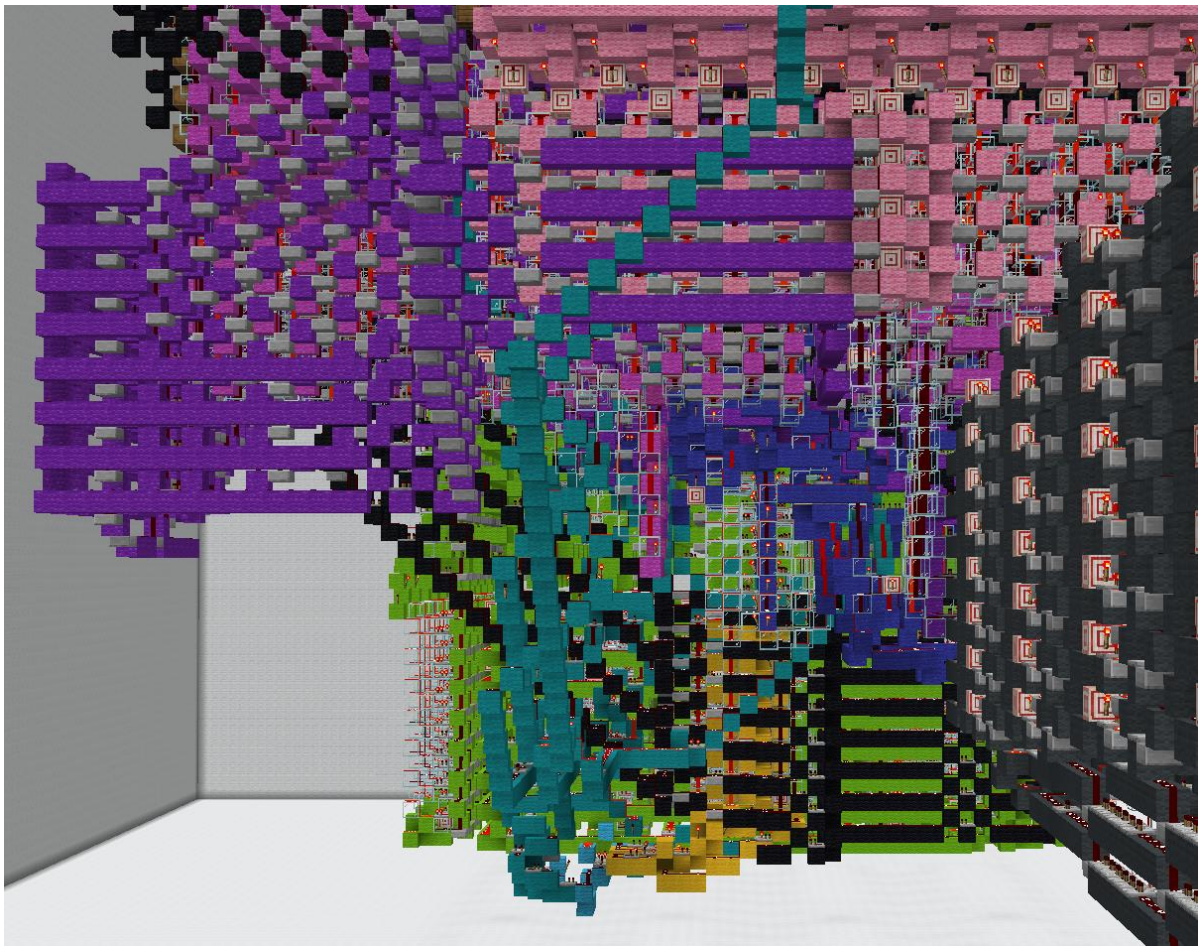
Za izvajanje logičnih operacij v registru za zastavice sem naredil posebno vezje, saj je primerjanje bitov veliko hitrejše, kot primerjanje bajtov s pomočjo ALE.

Največ problemov sem imel s povezavo krmilne enote z ROM-om (podobna je bila velikemu dekoderju), saj poleg tega da je bilo treba samo dekodirati instrukcijo, je bilo treba tudi dekodiran signal sinhronizirati, da je prispel pravi trenutek (npr. registra sta morala dobiti signal za branje podatka iz določenega mesta tako, medtem ko je moral ALE dobiti ukaz za seštevanje po eni sekundi oz. cca. 5-7 ciklih procesorja).

Na koncu sem dodal še podatkovna vodila (iz vsakega registra po 2, eno za branje in eno za pisanje), ki so napeljana izven procesorja. Zraven vsakega vodila sem napeljal še dodatno naslovno vodilo, po katerem pošiljam naslov naprave ali vezja, kateremu hočemo poslati oz. od katerega hočem prebrati podatke. Ti dve vodili sem dodal z mislijo na priključitev vhodno izhodnih naprav (npr. zaslon, tipkovnico…) in drugih vezij (npr. RAM, pretvornik iz binarnega številskega sestava v desetiškega…). S tem lahko dosežemo še večjo računsko moč tega procesorja.



Slika 34: CPE



Slika 35: CPE

Na zgornjih dveh slikah je prikazan celoten CPE. Siva barva prikazuje ALE, črna podatkovna vodila, roza registra za zastavice, magenta registra, vijolična vodila za naslove, zelena ROM, turkizna enobitne povezave (signal za aktivacijo določenega vezja, podatki iz registra za zastavice za ukaz JumpIf), modra glavni dekoder krmilne enote, rumena programski števec.

Zaključek

Pri izdelavi procesorja sem se veliko naučil o delovanju procesorja in delovanju redstona v Minecraftu. Pri tem sem naletel na veliko podobnih problemov kot v realnem življenju pri ustvarjanju vezij.

Trenutna različica procesorja ima naslednje lastnosti:

- 2x16 bytov registrov,
- 2x16 bytov registrov za zastavice,
- 4.667 Hz,
- v ROM-u je prostora za 256 16-bitnih instrukcij (glej tabelo spodaj),
- naslovno vodilo z dosegom 510 naslovov, namenjenih vhodnim operacijam,
- naslovno vodilo z dosegom 510 naslovov, namenjenih izhodnim operacijam.

Nabor instrukcij še ni dokončen: Za idejo, katere instrukcije so trenutno implementirane, pa sem zbral podatke v spodnji tabeli.

Tabela 1: Instrukcije moje implementacije CPE v Minecraftu

Mnemonik	Operacijska koda	Operandi	Opis instrukcije
NOP	0000	Brez	NO OPERATION Brez operacije.
JMPA	0001	Naslov registra (4b)	JumpA reg Prebere število iz bloka A in nanjo nastavi števec programa.
JMPB	0010	Naslov registra (4b)	JumpB reg Prebere število iz bloka B in nanjo nastavi števec programa.
JMPIAZ	0011	Naslov registra (4b), naslov registra za zastavice (4b)	JumpIfA reg, flag Prebere število iz bloka A in nanjo nastavi števec programa, če je prebran bit iz registra za zastavice enak 0.
JMPIBZ	0100	Naslov registra (4b), naslov registra za zastavice (4b)	JumpIfB reg, flag Prebere število iz bloka B in nanjo nastavi števec programa, če je prebran bit iz registra za zastavice enak 0.
OA	0101	Naslov registra (4b), naslov izhodne naprave (8b)	OutA reg, num Na izhodno podatkovno vodilo pošlje število iz bloka A. Zraven pošlje tudi število na naslovno vodilo.
OB	0110	Naslov registra (4b), naslov izhodne naprave (8b)	OutB reg, num Na izhodno podatkovno vodilo pošlje število iz bloka B. Zraven pošlje tudi število na naslovno vodilo.
IA	0111	Naslov registra (4b), naslov vhodne naprave (8b)	InA reg Prebere število iz vhodnega vodila v blok A. Zraven pošlje tudi število na naslovno vodilo.
IB	1000	Naslov registra (4b), naslov vhodne naprave (8b)	InB reg Prebere število iz vhodnega vodila v blok B. Zraven pošlje tudi število na naslovno vodilo.

CPB	1001	Vrednost 0 (4b), naslov registra (4b), naslov registra (4b)	CopyByte reg, reg Prestavi byte iz bloka A v blok B.
CPF	1010	Vrednost 0 (4b), naslov registra za zastavice (4b), naslov registra za zastavice (4b)	CopyBit flag, flag Prestavi bit iz registra za zastavice A v register za zastavice B.
ALEOP	1011	operacija ALE (4b), naslov prvega registra (4b) (uporabljen tudi za zastavice) naslov prvega registra (4b) (uporabljen tudi za zastavice)	ALEOper oper, reg, reg Naredi računsko operacijo nad izbranima bytoma iz bloka A in B z uporabo ALE. Shrani tudi 3 zastavice.
CMPOP	1100	Operacija komparatorja za zastavice (4b), naslov prvega registra za zastavice (4b), naslov drugega registra za zastavice (4b)	CmpOper oper, flag, flag Naredi bitno operacijo nad izbranima bitoma iz registra za za zastavice A in B s pomočjo komparatorja za bite.
SBFNOP	1101	Vrednost 0 (4b), Številka (8b)	SetByteForNextOP num Na izhodno podatkovno vodilo pošlje število (vodilo je povezano tudi z ALE)
SBFNOPR	1111	Brez	SetByteForNextOPRand ALE-ju pošlje naključno število za instrukcijo, ki se bo izvedla naslednji cikel.

Operacije v instrukciji ALEOP (štirje biti):

Koda operacije	Operacija
0001	Seštej s prenosom
0010	Zamik bitov v desno
0011	Zamik bitov v levo
0100	Bitni XNOR
0101	Bitni ALI
0110	Bitni IN
0111	Bitni XOR
1000	Seštej
1010	Odštej 1
1011	Odštej
1100	Množenje
1101	Deljenje

Potrditev ali zavrnitev hipotez

Uspelo mi je potrditi naslednje hipoteze:

- Potrjujem hipotezo, da imamo v Minecraftu tudi probleme s slabljenjem signala. Imel sem veliko problemov s šibkimi signali pri povezavah na daljše razdalje (npr. pri ROM, krmilna enota, povezave z vodili...). Imel sem jih tudi par pri dekoderju za ALE itd. Za

rešitev sem uporabljal redstone ojačevalnike, včasih pa sem kar izkoristil lastnosti samih redstone komponent (še posebej pri baklah).

- Prav tako potrjujem hipotezo, da poskušamo optimizirati vezja tudi v Minecraftu. Veliko časa sem porabil z optimizacijami pomnilnikov (registri, posebni registri, ROM), ALE, krmilne enote in tudi ostalih vezij. Optimiziral sem jih tako, kot se jih optimizira v digitalni elektroniki – z izbiro med alternativnimi izvedbami komponent, obliko v 3D prostoru in nenazadnje z matematično ustreznejšim pristopom k izvedbi.
- Hipotezo, da poskušamo narediti vezja čim manjša, sem potrdil, saj sem pomnilnike poskušal zelo optimizirati, da so čim bolj kompaktni in da zavzamejo čim manjšo prostornino, prav tako tudi ostala vezja. Z manjšimi vezji so bile tudi povezave med različnimi vezji lahko krajše, s tem sem se ponekod izognil tudi dodatnim ojačevalnikom.
- Potrjujem tudi hipotezo, da se poskušamo pri optimizaciji poslužiti potencialno delujočih trikov. Pri npr. paralelnem seštevalniku z uporabo cevovodov ali poenostavitvah vezij z matematičnimi ekvivalenti (De Morganov izrek ipd.). V Minecraftu se poslužujemo tudi veliko trikov glede moči signala, saj je moč signala od izhoda vsake komponente vedno največja, kar nam olajša načrtovanje razdalj med komponentami.
- Hipotezo, da v Minecraftu prihaja do podobnega problema organizacije povezav kot pri povezavah v digitalnih vezjih. S pametno organizacij povezav lahko v obeh primerih minimaliziramo potreben prostor. To sem potrdil pri poglavju Vezje za seštevanje, odštevanje in bitne operacije.
- Hipoteza, da v Minecraftu redstone komponente povzročajo zamudo v signalih, sem potrdil, saj ima vsaka redstone komponenta določen časovni zamik za aktivacijo. Tako kot lahko komponente v digitalni tehniki preidejo v nasičenje, se lahko podobno dogaja tudi v Minecraftu npr. pri redstone bakli. Tudi v Minecraftovih želimo, da signali odidejo na pot in da prispejo na cilj ob istem času (npr. pri vezju XOR).
- Hipotezo, da imajo lahko hrošče tako digitalna vezja kot tudi vezja v Minecraftu lahko potrdim, saj sem porabil kar nekaj časa z razhroščevanjem ALE in krmilne enote in trenutno v moji izvedbi vem za dva hrošča. Hrošče v digitalnih vezjih, kot so npr. centralne procesne enote, pa lahko najdemo popisane na veliko koncih spleta (npr. [5]).
- Prav tako potrjujem hipotezo, da poskušamo izdelati komponente na način, da jih lahko uporabimo v čim več vezjih. To lahko dokažem, ker sem npr. paralelni seštevalnik uporabil v kar treh različnih vezjih znotraj ALE (množenje, deljenje, seštevanje in odštevanje (eno samo vezje)).
- S potrjenimi vsemi zgornjimi hipotezami potrjujem tudi zadnjo: Minecraft je odličen učni pripomoček o tehniki izdelave digitalnih logičnih vezij, saj sem se veliko naučil o praktičnih pristopih v digitalni tehniki na nivoju same fizikalne zgradbe, kako delujejo procesne enote ter druge digitalne komponente.

Za konec

Z redstonom v Minecraftu se ukvarjam že približno 4 leta. To je bil moj drugi poskus izdelave tako kompleksnega logičnega vezja, kot je centralna procesna enota. Pri njeni izdelavi sem se zelo zabaval, saj me delovanje računalnikov zelo zanima. Na projektu sem delal zadnje tri mesece, skupaj približno 150 h. V bližnji prihodnosti načrtujem izdelavo uporabnih vhodnih in izhodnih naprav za ta procesor, ki bi služile kot uporabniški vmesnik in če se bodo zvezde pravilno poravnale morda še prevajalnik za zbirnik.

Viri

[1] Sandi Zakoršek. **Simulator logičnih vrat.** Univerza v Mariboru. Dostopno prek: https://dk.um.si/Dokument.php?id=135757&lang=slv (1. 3. 2024)

[2] mattbatwings. **8-bit Redstone Computer.** Dostopno prek: https://www.planetminecraft.com/project/8-bit-redstone-computer-5892975/ (1. 3. 2024)

[3] mattbatwings. **Addition | Logical Redstone #8.** Dostopno prek: https://youtu.be/BwY0Q7JzizQ (1. 3. 2024)

[4] mattbatwings. **Division | Logical Redstone #14.** Dostopno prek: https://youtu.be/QTO9msstzaw (1. 3. 2024)

[5] OSDev.org. **CPU Bugs.** Dostopno prek: https://wiki.osdev.org/CPU_Bugs (3. 3. 2024)