Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tim Thuma

MiMo model CPE

prva seminarska naloga pri predmetu organizacija računalnikov

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Robert Rozman

Ljubljana, junij 2025

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
smode	fpuop			fdreg				fsreg				ftreg			

Tabela 2: Struktura podatka, ki vključi izračun na FPU.

6 Šesta naloga

Navodilo Trenutno MiMo model nima strojne izvedbe računanja s plavajočo vejico. Dodajmo mu napravo, ki bo to podpirala.

Rešitev

FPU (floating point unit) bomo izvedli kot vhodno-izhodno napravo. Bolj programerju prijazna rešitev bi bila, da bi v zbirnik dodali ukaze, ki nadzirajo to enoto, toda v tem primeru sem se odločil, da bomo FPU nadzirali s pisanjem v rezerviran del pomnilnika. Ta del pomnilnika deluje kot pomensko preslikan (memory-mapped) pomnilnik. Torej ko bomo pisali na ustrezne naslove, se bodo zapisani podatki posredovali v FPU. Za delo s TTY rabimo samo en naslov, torej lahko del pomnilnika, kjer se naslovi začnejo z binarno `10` porabimo še za kaj drugega.

FPU ima 16 16-bitnih registrov. Te nam bodo omogočali računanje s polovično natančnostjo (half-precision), zato jih bomo imenovali H registri. Registri od H0 do H14 so bralno-pisalni, H15 in H16 pa sta samo pisalna. Več o tem bom razložil kasneje.

Po dva H registra se lahko združita v S registre, ki jih bomo uporabljali za računanje z enojno natančnostjo (single precision). V S registre ne moremo direktno pisati, ampak moramo vedno obe polovici (oba H registra) napolniti posebej. Prav tako jih ne moremo direktno brati. Direktno lahko beremo le rezultat operacij, ki uporabljajo S registre.

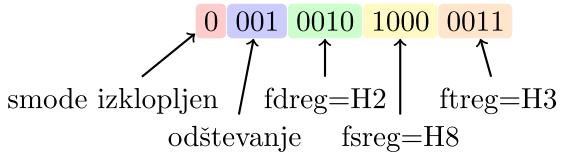
Pisanje v H registre poteka s pisanjem na naslove od 32768 do 32783, ki se preslikajo na registre H0 do H15. Številka H registra v katerega želimo pisati se torej vzame iz najnižjih 5 bitov naslova, ki ima največja bita enaka `10`. Podobno se tudi branje H registrov izvede z izvedbo ukazov za branje na teh enakih naslovih (z izjemo H14 in H15).

Ko želimo v FPU nekaj izračunati, moramo pisati na naslov 32784. Vsebina podatka, ki ga zapišemo na ta naslov je strukturirana na način kot je prikazan v Tabeli 2. `fdreg`, `fsreg` in `ftreg` imajo tu podoben pomen kot `dreg`, `sreg` in `treg` pri zbirniških ukazih. Npr. če bomo v del za `fsreg` vpisali `0101`, se bo za izvorni register uporabil H5. Bit `smode` je prižgan, če želimo namesto H registrov uporabiti S registre in računati z enojno namesto s polovično natančnostjo. `fpuop` nam pove, katero operacijo bomo izvedli nad vhodnimi registri. Te si sledijo tako:

- 0 = seštevanje
- 1 = odštevanje

- 2 = množenje
- 3 = deljenje
- 4 = „multiply-accumulate“

Poglejmo si to na konkretnem primeru:



Pri operacijah v H načinu se lahko na izhodu prižgeta tudi dve zastavici `zero` in `NaNerror`. Prva se prižge, ko je rezultat operacije enak nič, druga pa, ko je pri operaciji prišlo do napake in rezultat ni veljavno število (npr. če smo delili z nič).

Pri uporabi S načina imamo na voljo še eno dodatno operacijo, ki izračuna inverz korena po načinu, ki je uporabljen v igri Quake III Arena (1999).

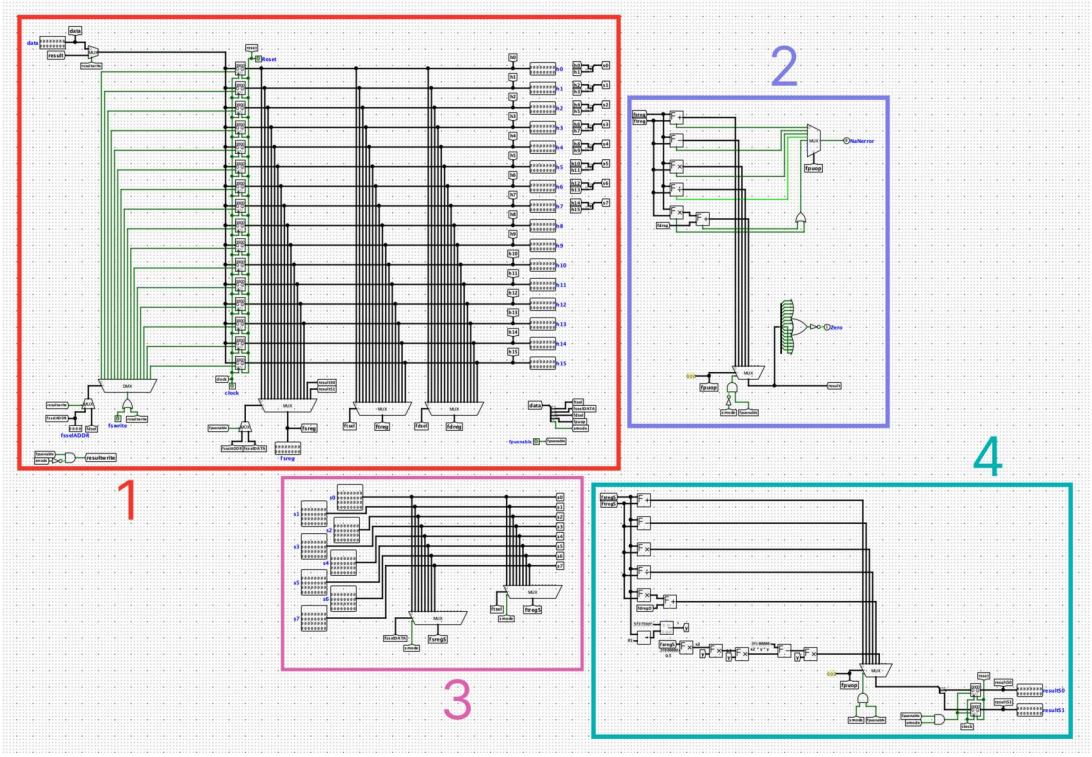
Za razliko od H načina se v S načinu rezultat ne zapiše v register, ki ga navedemo kot `fdreg`, ampak se shrani v dva posebna registra za rezultat. Alternativno bi lahko rezultat pisali v H registre, ampak to bi bilo štorasto. Registroma, ki hranita rezultat S mode operacij, rečemo `resultS0` in `resultS1`, beremo pa jiju lahko z branjem na naslovih 32782 in 32783. Ta dva naslova se uporabljata tako za pisanje v registra H14 in H15, kot tudi za branje registrov `resultS0` in `resultS1`.

Poglejmo si še kako je opisani FPU izveden v vezju. Celotno vezje lahko razdelimo na štiri glavne dele, kot je prikazano na Sliki 17. Pri tem so oznake delov naslednje:

1. Registrska enota za H registre.
2. Aritmetična enota za H način.
3. Enota za izbiranje S registrov.
4. Aritmetična enota za S način.

Začnimo po vrsti z registrsko enoto za H registre. Ta je bolj podrobno prikazana na Sliki 18. V levem zgornjem kotu vidimo podatke, ki se lahko zapišejo v registre. Te so bolj podrobno prikazani na Sliki 19. Podatki lahko pridejo kot vhod `data`. Sem podatki pridejo iz podatkovnega vodila, ko pišemo z ukazi `sw` in podobnimi ukazi. Ko pa sprožimo računanje, se rezultat iz tunela `result` zapiše v register, ki smo ga izbrali kot `fdreg`.

Ko uporabimo ukaz `sw` za pisanje v FPU registre, se indeks H registra, v katerega pišemo, dekodira kar iz naslova in pošlje kot `fsselADDR`. Ob tem se tudi prižge `fswrite` signal. Ko pa pišemo iz rezultata FPU operacije v H register, se prižge `resultwrite` signal. Ob tem moramo imeti v `fdsel` zapisan indeks registra, v katerega bomo pisali (destination register). `resultwrite` se vklopi, kadar pišemo na naslov 32784 (naslov za izračun) v H načinu, saj se le takrat rezultat vpisuje nazaj v `fdreg`. Logika za izbiro



Slika 17: Celotno vezje FPU z označenimi deli.

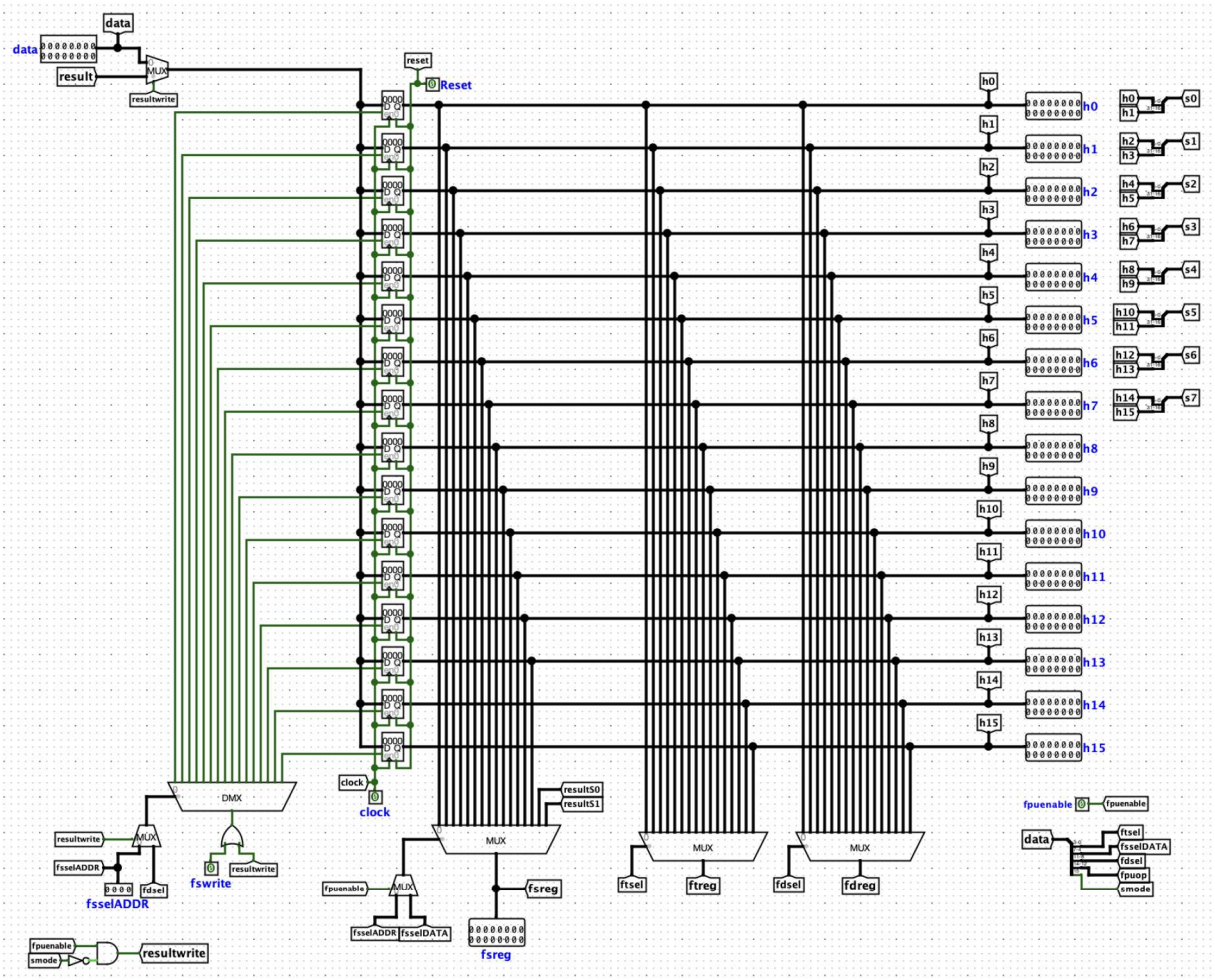
registra, v katerega pišemo, je prikazana na Sliki 20. Vezje, ki nastavi `resultwrite` signal je prikazano na Sliki 21.

Ko pišemo na naslov 32784, se v FPU pošlje `fpuenabled` signal, hkrati pa se podatki iz podatkovnega vodila dekodirajo po pravilu, ki sem ga opisal prej. Vezje za dekodiranje FPU ukaza je prikazano na Sliki 22. Tu vidimo, da se za ločevanje, ali `fssel` dekodiramo iz naslova (v primeru direktnega pisanja v H register) ali iz podatkovnega vodila (v primeru sprožitve FPU operacije), uporabljata `fsselADDR` in `fsselDATA` tunela.

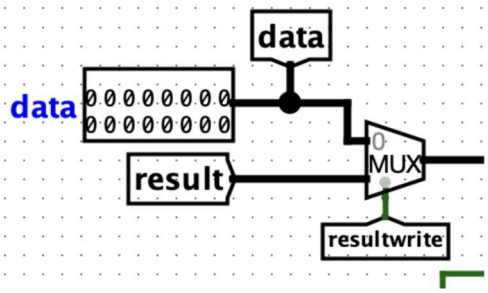
Izbiranje ustreznih registrov, ki se bodo poslali kot vhod v aritmetično enoto je prikazano na Sliki 23. Tu so povezave, ki pridejo v multiplekserje povezane z zaporednimi H registri. Pri `fsreg` vidimo, da lahko selektor pride iz naslova ali iz podatkovnega vodila. `fsselDATA` se bo uporabil, kadar je vklopljen `fpuenable`, torej kadar pišemo na naslov 32784. Tu vidimo tudi, da H14 in H15 registrov ne moremo brati v `fsreg`, saj te dve mesti zasedata registra rezultatov v S načinu. Zato sta ta dva registra nedosegljiva za branje izven FPU.

Poglejmo si enoto za aritmetiko v H načinu, kot je prikazana na Sliki 24. Na dnu vidimo, da se rezultat tu izračuna samo, kadar smo v H načinu in imamo prižgan `fpuenable` signal. Edina netrivialna stvar pri tem delu je operacija multiply-accumulate. Ta vrednosti v `fdreg` registru prišteje zmnožek `fsreg` in `ftreg` registrov. To je uporabno pri množenju matrik.

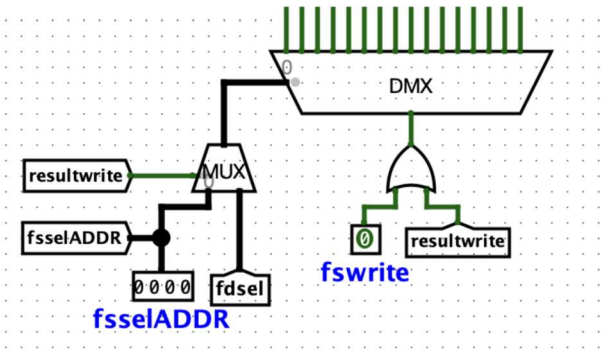
Na Sliki 25 je prikazano, kako se H registri združijo v S registre. Vidimo, da S registri



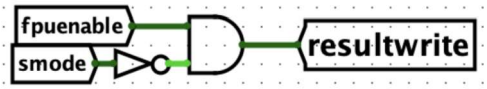
Slika 18: Registrska enota za H registre.



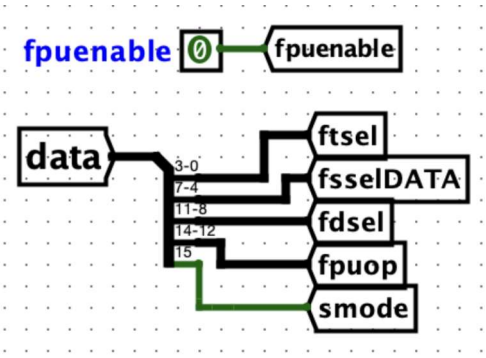
Slika 19: Vhodni podatki v registrsko enoto za H registre.



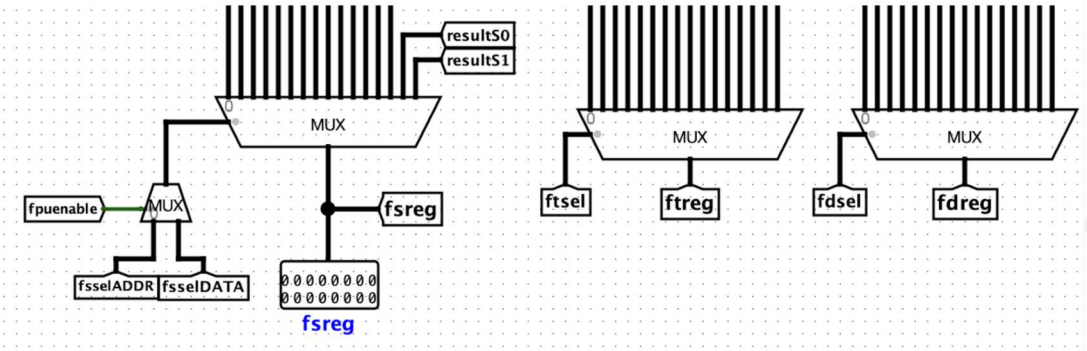
Slika 20: Selekcija registra za pisanje v H registrski enoti.



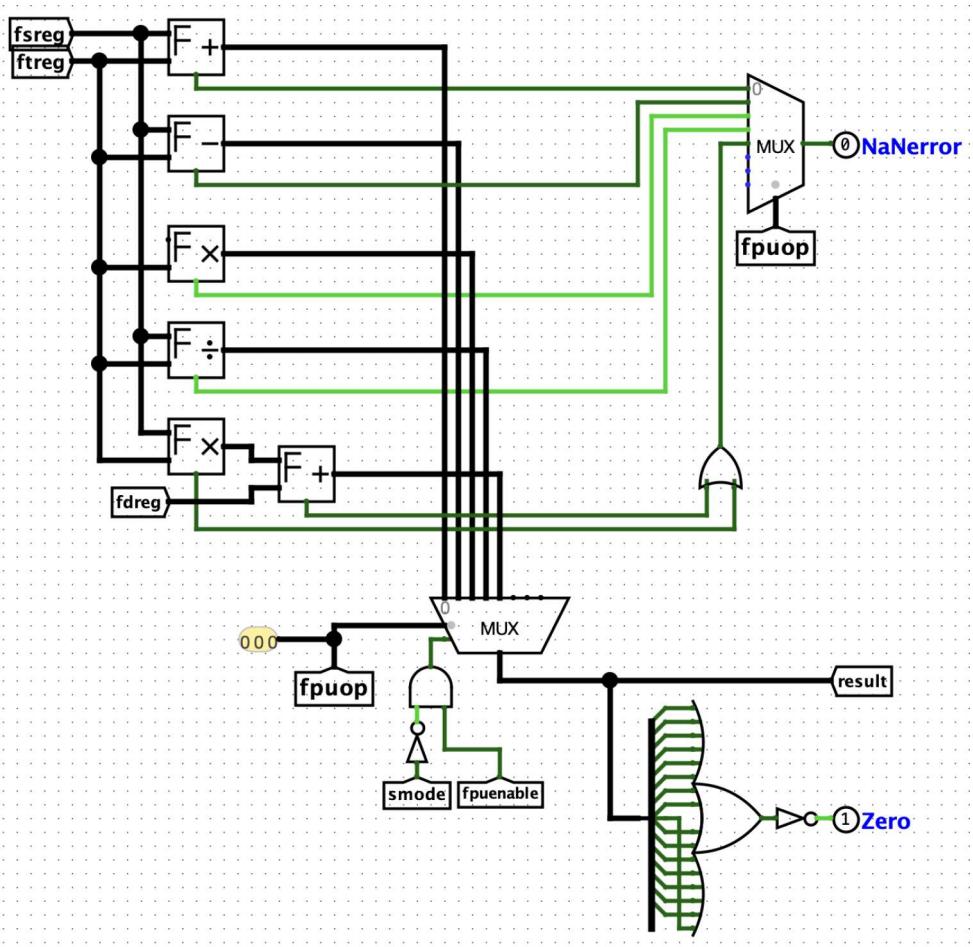
Slika 21: Vezje za vklop pisanja rezultata FPU.



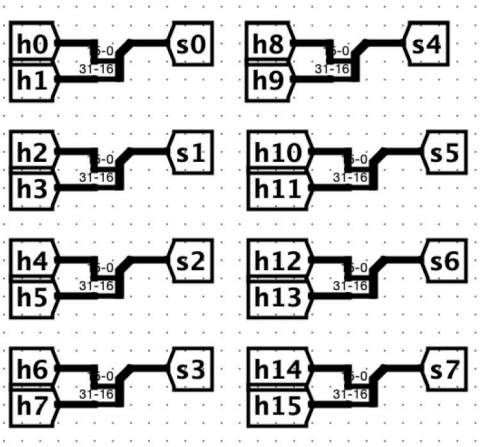
Slika 22: Dekodiranje FPU ukaza.



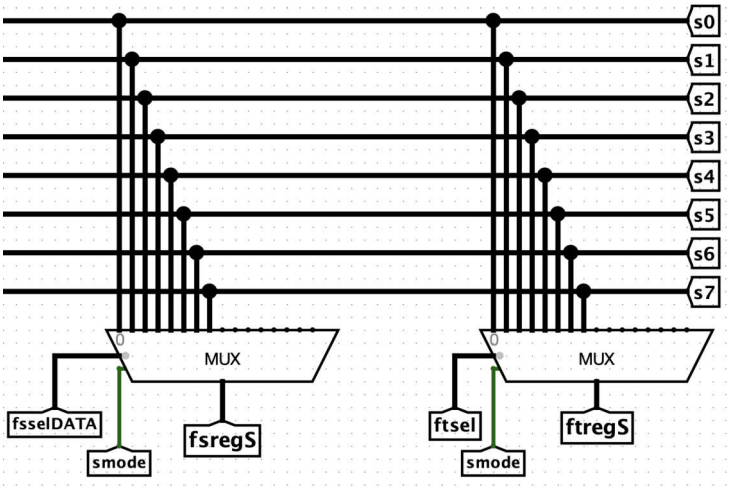
Slika 23: Izbor vhodnih registrov za aritmetično enoto.



Slika 24: Enota za aritmetiko v H načinu.



Slika 25: Združevanje H registrov v S registre.

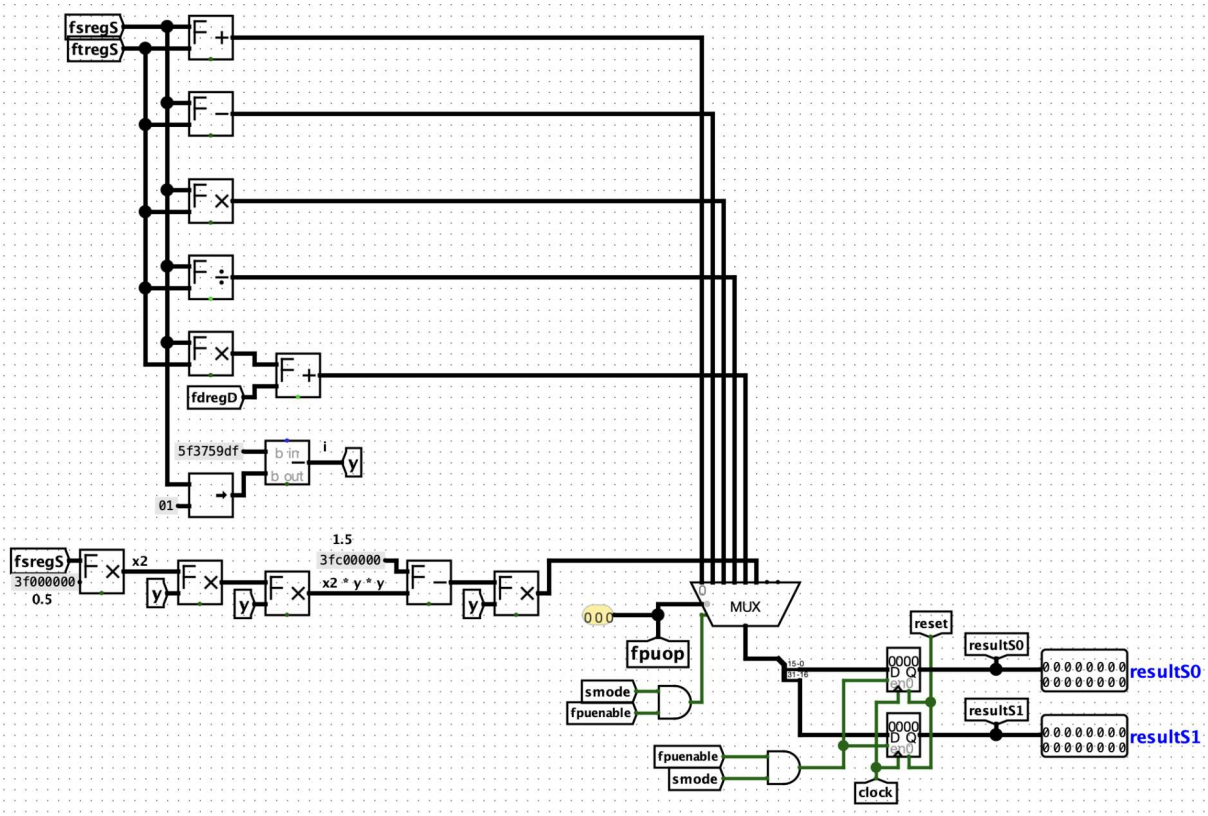


Slika 26: Izbiranje ustreznih S registrov.

niso zares registri, ampak so samo povezave po dveh H registrov. Izbira ustreznega S registra glede na podatke iz podatkovnega vodila je podobna izbiri H registrov. To je prikazano na Sliki 26. Vidimo, da je tu za selektor `fsreg` vedno uporabljen `fsselDATA`, sej se S registrov ne da direktno brati oz. pisati vanje.

Aritmetična enota za S način je zelo podobna tisti za H način. Ta del FPU je prikazan na Sliki 27. Desno spodaj vidimo, da je, podobno kot aritmetična enota za H način, tudi ta prižgana samo, kadar sta vklopljena `fpuenable` in `smode` signala. Takrat je tudi omogočeno pisanje v `resultS0` in `resultS1` registra. Šesta operacija v tej enoti nam izračuna inverza kvadratnega korena števila ($\frac{1}{\sqrt{x}}$). V bistvu gre za strojno izvedbo algoritma, ki je uporabljen v igri Quake III Arena [1]. Njegov zapis v programskem jeziku C je prikazan na Sliki 28. V našem primeru je algoritem izveden z eno iteracijo Newtonove metode. Algoritem je zasnovan za delo z 32-bitnimi števili, zato ga je mogoče izvajati le v S načinu.

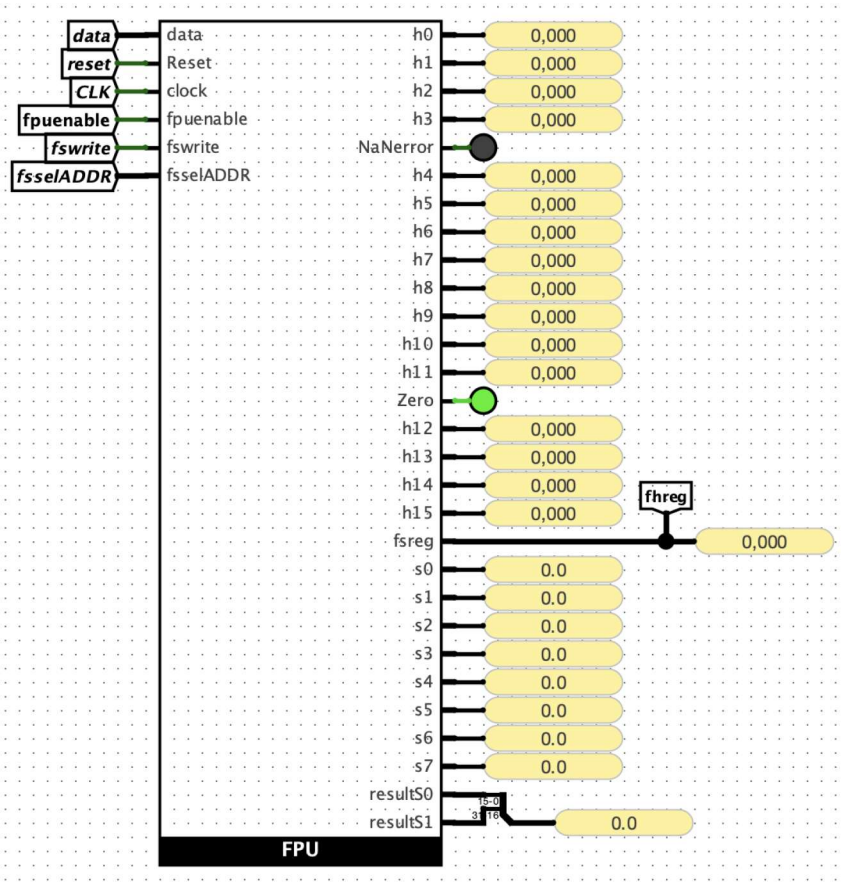
Poglejmo si še, kako dodamo FPU v MiMo model. To je prikazano na Sliki 29. Da



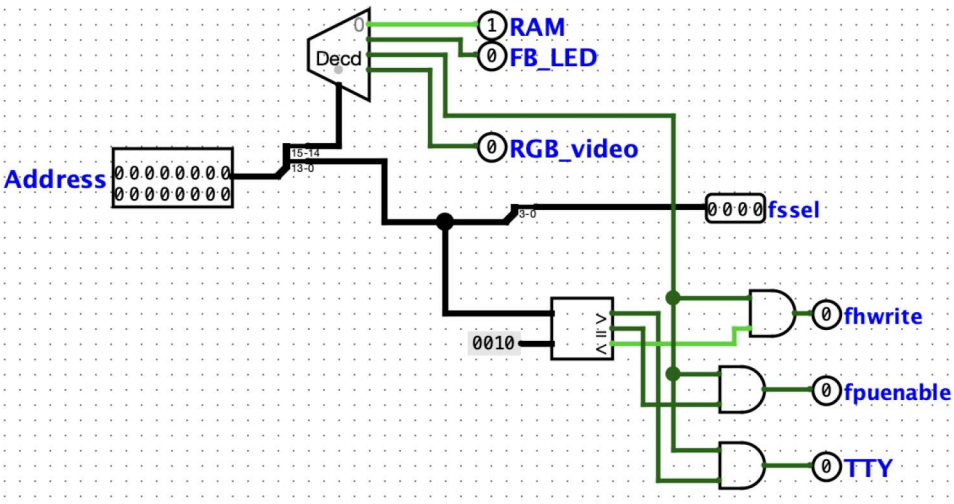
Slika 27: Aritmetična enota za S način.

```c
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

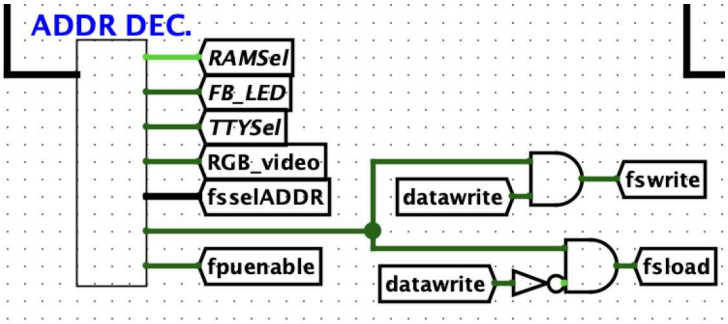
Slika 28: Koda za hitri izračun inverza kvadratnega korena iz igre Quake III Arena.



Slika 29: FPU enota, ko jo vključimo v MiMo model.



Slika 30: Popravljen naslovni dekodirnik, ki omogoča uporabo FPU.



Slika 31: Uporaba signala `datawrite` za ugotavljanje ali pišemo ali beremo iz H registra.

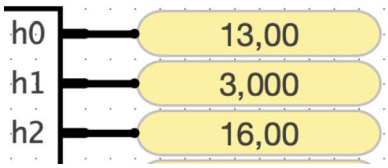
se `fpuenable`, `fswrite` in `fsselADDR` pravilno pošljejo v FPU, jih moramo dekodirati z naslovnim dekoderjem. Popravljen naslovni dekoder je prikazan na Sliki 30. Za delo s FPU smo porabili del pomnilnika, ki je bil prej rezerviran samo za TTY. TTY se sedaj lahko uporablja samo na naslovih, ki so večji od `0b1000000000010000` oz. `32785`. Iz najnižjih treh bitov naslova se prebere `fssel`. Če sta najvišja bita naslova enaka `10` in pišemo na naslove med 32768 in 32783, se bo vklopil `hwrite`, s katerim pišemo v H registre. Če pa pišemo na naslov 32784, se bo prižgal `fpuenable` in FPU bo šel v način računanja. To nam zagotovi komparator v spodnjem delu naslovnega dekodirnika na Sliki 30. Da vemo, ali gre za branje ali pisanje H registra, pogledamo še vrednost kontrolnega signala `datawrite`. Spremenjena logika je prikazana na Sliki 31.

6.1 Testni programi za FPU

Poglejmo si nekaj testnih programov z uporabo FPU. Na Sliki 32 je prikazana preprosta koda za seštevanje dveh decimalnih števil. Najprej naložimo vrednosti 13 in 3 v decimalnem zapisu po standardu IEEE 754, ki je že pretvorjen v ustrezne celoštevilske

```
main: li r0, 19072 # 13
li  r1, 16896 # 3
sw  r0, 32768
sw  r1, 32769

li  r1, 201 # 0|000|0010|0000|0001
sw  r1, 32784
lw r2, 32770
```

Slika 32: Testni program za seštevanje dveh decimalnih števil.



(a) H registri FPU enote ob končanju testnega programa za seštevanje.

(b) Rezultat seštevanja decimalnih števil v R2 registru.

Slika 33: Rezultati testnega programa za seštevanje.

vrednosti. Nato jiju shranimo v registra H0 in H1. Potem izvedemo operacijo seštevanja v H načinu, kjer se rezultat shrani v H2. Nato H2 prebermo v R2. Stanje H registrov po zaključku programa je prikazano na Sliki 33a, rezultat v registru R2 pa je prikazan na Sliki 33b.

V tem primeru smo morali te dve števili sami pretvoriti v IEEE 754 zapis, kar je rahlo štorasto. Prav tako trenutno zbirnik nima možnosti podajanja takojšnjih operandov v binarnem zapisu, zaradi česar si moramo najprej napisati ukaz za FPU v binarnem zapisu in ga pretvoriti v desetiški zapis. Za lažje delo z decimalnimi števili nadgradimo zbirnik. Za to bomo dodali novo podrutino `convert_number`, ki pretvarja števila iz šestnajstiškega, osmiškega in dvojiškega sistema ter iz zapisa s plavajočo vejico v zapis v desetiškem sistemu. Ta podrutina je prikazana na Sliki 34. Uporabili jo bomo povsod, kjer se takojšnji operand pretvarja v desetiški sestav. Tako dobimo spremembo, ki je prikazana na Sliki 35 in podobno naredimo še na ostalih mestih. Za zaznavo števil s plavajočo vejico v takojšnjih operandih bomo pogledali, ali operand vsebuje decimalno piko, in ga v tem primeru pretvorili po IEEE 754 algoritmu. Najvišji bit predstavlja predznak, naslednjih pet eksponent in nato deset bitov mantiso (decimalni del). Z uporabo Perlovih vgrajenih funkcij lahko število pretvorimo v 32-bitni zapis s plavajočo vejico, nato pa moramo to ročno pretvoriti na 16-bitov za računanje s polovično natančnostjo. Z vrstico, ki je označena na Sliki 34, omogočimo pretvorbo števil iz zapisa s plavajočo vejico. Dodana podrutina `float_to_ieee754_16` je prikazana na Sliki 36. V nadaljevanju si bomo

```
sub convert_number {
    my ($str) = @_;

    return hex($str) if ($str =~ m{^0x});
    return oct($str) if ($str =~ m{^0b});
    return oct(substr($str, 2)) if ($str =~ m{^0o});
    return float_to_ieee754_16($str) if $str =~ m{^-?\d+\.\d+$};

    return $str;
}
```

Slika 34: Podrutina za pretvorbo števila v desetiški sistem. Označeni del podano število pretvori po IEEE 754 standardu, če to vsebuje decimalno piko.

```
if ($atype eq 'i') {
    $arg= hex($arg) if ($arg =~ m{^0x});
    $PC++; $Mcode[$PC]= $arg; $Ltype[$PC]=1;
}
```

```
if ($atype eq 'i') {
    $arg= convert_number($arg);
    $PC++; $Mcode[$PC]= $arg; $Ltype[$PC]=1;
}
```

Slika 35: Sprememba zbirnika s katero omogočimo pretvarjanje števil.

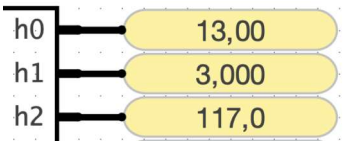
pogledali še nekaj primerov uporabe FPU po dodanih spremembah.

```perl
sub float_to_ieee754_16 {
    my ($float) = @_;

    # get 32-bit float in binary
    my $binary = unpack("L", pack("f", $float));

    # Extract sign (1 bit), exponent (5 bits), and fraction (10 bits)
    my $sign = ($binary >> 31) & 0x1;
    my $exp  = (($binary >> 23) & 0xFF) - 127 + 15;  # Adjust bias
    my $frac = ($binary >> 13) & 0x3FF;  # Get only top 10 bits of fraction

    if ($exp < 0) {
        # Subnormal number (set exponent to zero and shift fraction)
        $frac = ($binary >> (23 - 10 + $exp)) & 0x3FF;
        $exp = 0;
    } elsif ($exp > 31) {
        # Overflow (return infinity)
        return ($sign << 15) | (31 << 10);
    }

    return ($sign << 15) | ($exp << 10) | $frac;
}
```

Slika 36: Dodana podrutina `float_to_ieee754_16`.

```
main: li r0, 19072 # 13
li  r1, 16896 # 3
sw  r0, 32768
sw  r1, 32769

li  r1, 16897 # 0|100|0010|0000|0001
sw  r1, 32784
sw  r1, 32784
sw  r1, 32784
lw r2, 32770
```

(a) Koda primera seštevanje 3.0 + 13.0.



(b) Rezultat seštevanja 3.0 + 13.0.

Slika 37: Primer in rezultat operacije seštevanja 3.0 + 13.0.

6.1.1 Prvi primer

Ta primer je podoben prvemu, le da smo operacijo seštevanja zamenjali za „multiply accumulate“ ki jo izvedemo trikrat. Rezultat je torej enak $(H0 \cdot H1)+(H0 \cdot H1)+(H0 \cdot H1)$, kar je enako $3 \cdot (3 \cdot 13) = 117$. Koda in rezultat tega primera sta prikazana na Sliki 37.

6.1.2 Drugi primer

Preizkusimo razliko v natančnosti med načinom s polovično in z enojno natančnostjo. Na Sliki 38 vidimo izračun izraza $0.1 + 0.3$ s polovično in enojno natančnostjo. Način z enojno natančnostjo ima veliko manjšo napako od načina s polovično natančnostjo.

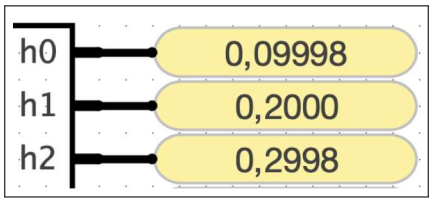
6.1.3 Tretji primer

V FPU imamo vgrajeno operacijo „multiply accumulate“, ki številu v `fdreg` prišteje zmnožek dveh števil. To nam poenostavi matrično množenje. Na enačbi (1) je prikazan primer množenja dveh matrik, katerega rezultat je rotacija točke v 2D prostoru. Vizualizacija te rotacije je vidna na Sliki 40. Rezultat, ki ga dobimo, je zaradi majhne natančnosti napačen na tretji decimalki. V registre od H0 do H8 smo zapisali rotacijsko matriko, v registre od H9 do H11 pa točko. Rezultat množenja se shrani v registre od H12 do H14. Množenje z uporabo „multiply accumulate“ lahko naredimo v devetih računskih korakih, namesto v 15 korakih, ki bi jih potrebovali z ročnim množenjem in seštevanjem.

$$\begin{bmatrix} \cos(30°) & -\sin(30°) & 0 \\ \sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1.5 \\ 3.5 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.451 \\ 3.781 \\ 1 \end{bmatrix} \tag{1}$$

```
main: li r0, 0b0010111001100110 # 0.1
li  r1, 0b0011001001100110 # 0.2
sw  r0, 32768
sw  r1, 32769

li  r1, 0b0000001000000001
sw  r1, 32784
```

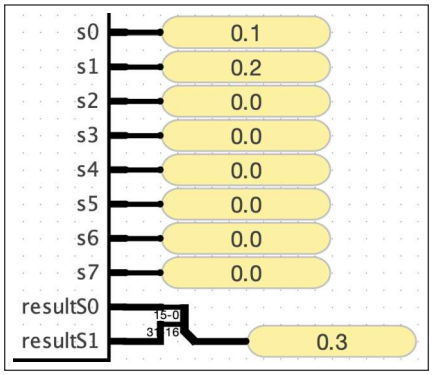
(a) Koda za izračun izraza $0.1 + 0.2$ s polovično natančnostjo.



(b) Rezultat izračuna izraza $0.1 + 0.2$ s polovično natančnostjo. Rezultat je napačen na četrti decimalki zaradi napake pri pretvorbi.

```
main: li r0, 0b1100110011001101 # 0.1 lower bits
      li r1, 0b0011110111001100 # 0.1 upper bits
li r2, 0b1100110011001101 # 0.2
li r3, 0b0011111001001100 # 0.2

sw  r0, 32768
sw  r1, 32769
sw  r2, 32770
sw  r3, 32771

li r0, 0b1000000000010000
sw r0, 32784

lw r0, 32782
lw r1, 32783
```

(c) Koda za izračun izraza $0.1 + 0.2$ z enojno natančnostjo.

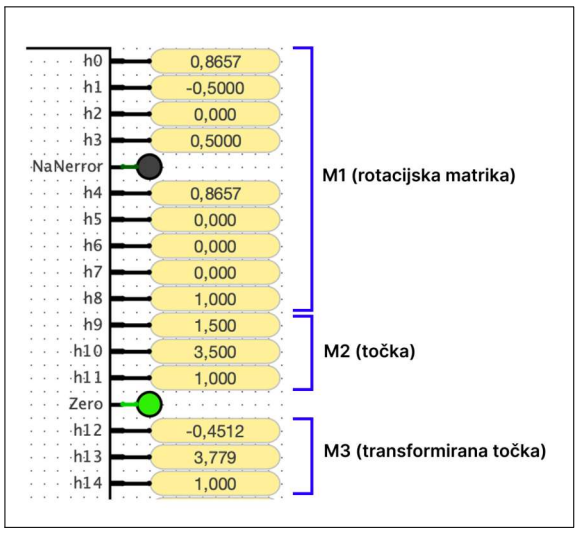


(d) Rezultat izračuna izraza $0.1 + 0.2$ z enojno natančnostjo. Rezultat je dovolj točen, da Logisim ne zazna napake, ki se zgodi na osmi decimalki.

Slika 38: Programa in rezultata računanja s polovično in z enojno natančnostjo.

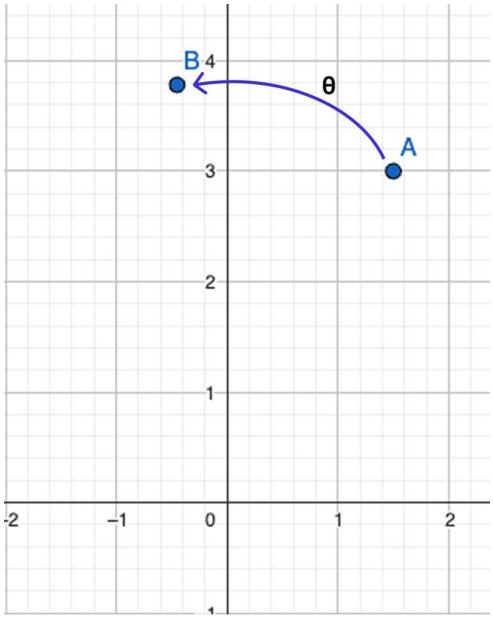
```
main: li r0, 0.866 # cos30
li  r1, -0.5 # -sin30
li r2, 0
li r3, 0.5 # sin30

li r4, 1.5 # 1.5
li r5, 3.5 # 3.5
li r6, 1.0 # 1

sw  r0, 32768
sw  r1, 32769
sw  r2, 32770
sw  r3, 32771
sw  r0, 32772
sw  r2, 32773
sw  r2, 32774
sw  r2, 32775
sw  r6, 32776

sw  r4, 32777
sw  r5, 32778
sw  r6, 32779


li  r1, 0b00101110000001001 # 0|010|1100|0000|1001
li  r2, 0b00100110000011010 # 0|100|1100|0001|1010
li  r3, 0b00100110000101011 # 0|100|1100|0010|1011
sw  r1, 32784
sw  r2, 32784
sw  r3, 32784

li  r1, 0b00101110100111001 # 0|010|1101|0011|1001
li  r2, 0b00100110101001010 # 0|100|1101|0100|1010
li  r3, 0b00100110101011011 # 0|100|1101|0101|1011
sw  r1, 32784
sw  r2, 32784
sw  r3, 32784

li  r1, 0b00100000001101001 # 0|010|0000|0110|1001
li  r2, 0b00100000001111010 # 0|100|0000|0111|1010
li  r3, 0b00100000010001011 # 0|100|0000|1000|1011
sw  r1, 32784
sw  r2, 32784
sw  r3, 32784
```

(a) Koda za matrično množenje.



(b) Rezultat matričnega množenja.

Slika 39: Realizacija matričnega množenja iz enačbe (1) in Slike 40 s FPU enoto. Rezultat množenja je rotacija točke v 2D prostoru.



Slika 40: Rotacija točke $[1.5, 3.5]$ za $30°$ v 2D prostoru.

7 Zaključki in možne nadgradnje

Vsa koda in Logisim vezje je na voljo na Github repozitoriju [2]. Projekt bi lahko še izboljšal tako, da bi ukaze za FPU dodal direktno v zbirnik, namesto da bi zanje uporabljal pomensko preslikan pomnilnik. Problem tega je, da bi moral za to razširiti kontrolno vodilo, kar lahko prinese nepričakovane zaplete. Poleg tega bi v zbirnik lahko dodal zmožnost izračuna trigonometričnih funkcij v takojšnjih operandih (npr. `li r0, sin(30)`) in pogostih decimalnih konstant (npr. π). Zbirnik bi lahko nadgradil tudi tako, da bi dodal za `pcsel` kontrolni signal dodal možnost `pcsel=databus`. S tem bi se lahko izognil kopiranju skočnih naslovov v takojšnji register pri ukazu `rts`.

Literatura

[1] Wikipedia. Fast inverse square root — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Fast%20inverse%20square%20root&oldid=1280110310, 2025. [Online; dostopano 24. marec 2025].

[2] Tim Thuma, Robert Rozman. MiMo Student Release. https://github.com/tthuma1/MiMo_Student_Release_Homework, 2025. [Online; dostopano 16. junij 2025].