# Programiranje 2, letnji semestar 2023/4

# Osnovna struktura C/C++ programa

Stefan Nikolić

Prirodno-matematički fakultet, Novi Sad
26.02.2024.

Na prošlom predavanju smo pokušali da ilustrujemo zašto je za matematiku važno programiranje

# Jedan primer u finansijskoj matematici

Bloomberg

Baruch College campus in New York. Photographer: Victor J. Blue/Bloomberg

**School of Quant: At $29,000, a Public NYC College Outclasses Princeton**

Here are the hard numbers: An 18-month Baruch MFE costs a New York City resident about $29,000. A similar degree from MIT costs four times that - and starting salaries for those grads are about a fifth lower. On average, fresh Baruch MFEs can expect almost $170,000, before bonus, according to QuantNet. (Baruch puts that figure closer to $220,000.)

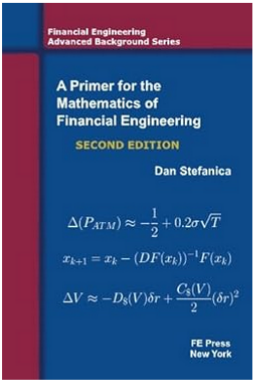
# Jedan primer u finansijskoj matematici

amazon Deliver to Serbia Books Search Amazon EN

All Today's Deals Customer Service Registry Gift Cards Sell

Books Advanced Search New Releases Best Sellers & More Amazon Book Clubs Children's Books Textbooks Best Books of the Month Best Books

Books › Business & Money › Job Hunting & Careers




## A Primer For The Mathematics Of Financial Engineering, Second Edition (Financial Engineering Advanced Background Series)

by Dan Stefanica (Author)

4.5 ★★★★½ 73 ratings

Book 1 of 4: Financial Engineering Advanced Background Series See all formats and editions

**Reviews** for "A Primer for the Mathematics of Financial Engineering", First Edition:

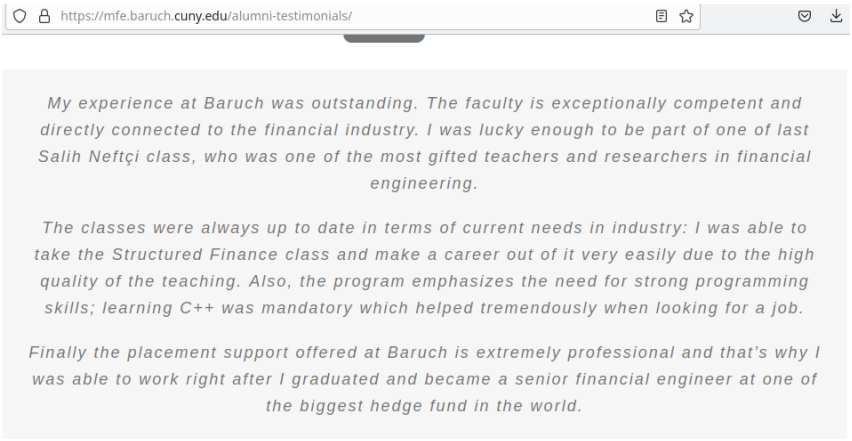
``One of the hottest degrees on today's campus is a Masters in Financial Engineering. Whether you need to retrieve hallowed memories or just want to familiarize yourself with the mathematics underlying this degree, this unique book offers a terrific return on investment."

--Peter Carr, PhD

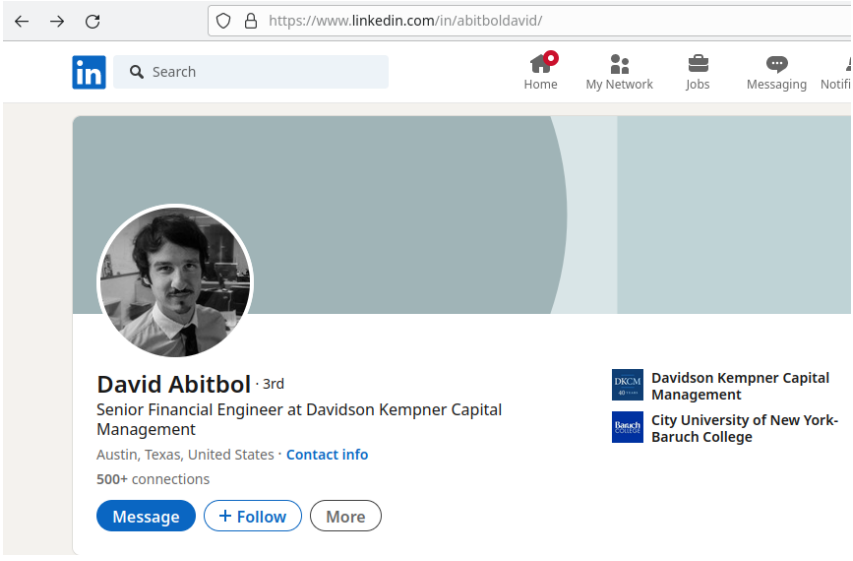
Global Head of Modeling, Morgan Stanley; Director of the Masters Program in Mathematical Finance, Courant Institute, NYU

``This is the book I always recommend to people who ask about their mathematics before

# Jedan primer u finansijskoj matematici



# Jedan primer u finansijskoj matematici



# Jedan primer u finansijskoj matematici

https://en.wikipedia.org/wiki/Davidson_Kempner_Capital_Management 120%

## Davidson Kempner Capital Management

1 language

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

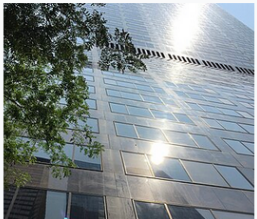
**Davidson Kempner Capital Management LP** (often known by the short form **Davidson Kempner**) is a global institutional alternative investment management firm with over $36 billion in assets under management.[5][6] Davidson Kempner is headquartered in New York City, with additional offices in London, Hong Kong, Dublin, Philadelphia, Shenzhen and Mumbai.[7][8][9] The firm is led by Anthony A. Yoseloff who serves as Executive Managing Member and Chief Investment Officer.[10]

As of June 30, 2022, Davidson Kempner was ranked as the 8th largest hedge fund in the world.[11]

Davidson Kempner has approximately 500 employees in the firm's seven offices.[4] The firm is headquartered at 520 Madison Avenue in New York, New York.[5][12]

**Davidson Kempner Capital Management LP**

DavidsonKempner
Capital Management LP



# Jedan primer u finansijskoj matematici

# Jedan primer u finansijskoj matematici

https://www.dmi.uns.ac.rs/wp-content/uploads/2022/12/Maxeler-is-hiring-C.pdf

1 of 1 — + Automatic Zoom

MAXELER
a groq company

## MAXELER IS HIRING!

We are seeking people who care about how computers work with skills in science, programming and High-Performance Computing, to join our team in **Belgrade, Novi Sad** or **Kragujevac**.

To apply, email your CV to **hiring.belgrade@maxeler.com**.


**JOIN THE DATAFLOW COMPUTING FUTURE!**

# Zadaci za danas

1. Osnovna struktura C/C++ programa
2. Kako radi računar?
3. Šta radi kompajler?

# Zašto je bitno da znamo kako radi računar?

- Da bismo što manje toga morali da učimo napamet
  (ako budemo razumeli kako radi računar, nećemo znati samo kako izgledaju programi u C/C++-u, već i zašto tako izgledaju)
- Jer ćemo samo ako razumemo kako radi računar moći da pišemo programe visokih performansi, što je za matematičke probleme jako bitno

# Nekoliko napomena

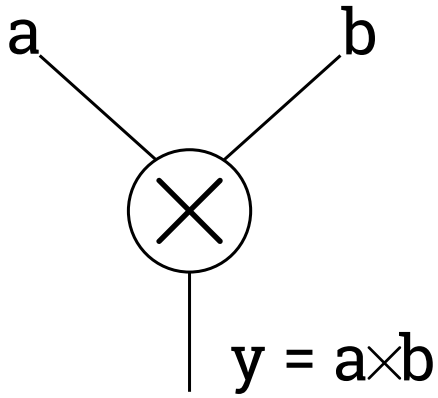
- Obradićemo samo osnovne principe, ali ćemo ih izvesti od početne intuicije
- Na kraju ćemo navesti preporučenu literaturu za one koji budu želeli da znaju više
- Ovo neće ići na usmeni ispit, ali će omogućiti lakše i dublje razumevanje daljeg gradiva

# Problem sa prethodnog predavanja: Ojlerova hipoteza o zbirovima stepena

1. Izlistavamo redom (leksikografski) kombinacije $n-1$ brojeva iz $[1, k]$, $k \in \mathbb{N}$, $a_1, a_2, ..., a_{n-1}$
2. Za svaku kombinaciju izračunamo $S = \sum_{i=1}^{n-1} a_i^n$
3. Pretpostavimo da je $S$ oblika $b^n$
4. Izračunamo $b$ kao $\lfloor \sqrt[n]{S} \rfloor$
5. Ako je $S = b^n$, našli smo kontraprimer

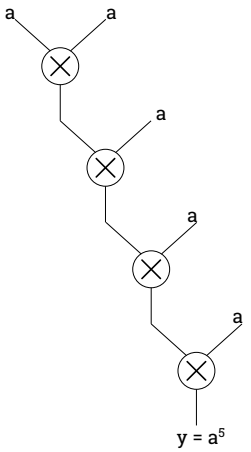
Koja operacija je odnosila najviše vremena?

# Množač

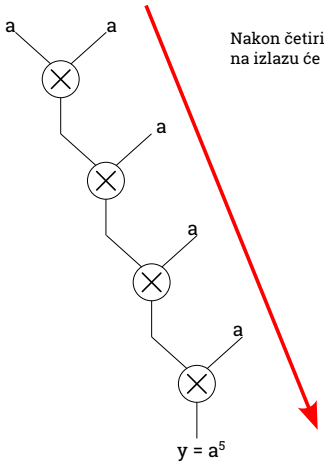



Možemo li da izračunamo $a^5$ pomoću 4 množača?

# Stepenovanje množačima

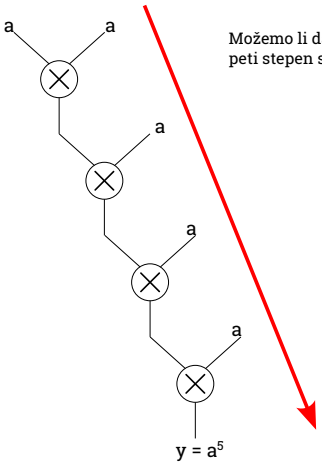


# Stepenovanje množačima



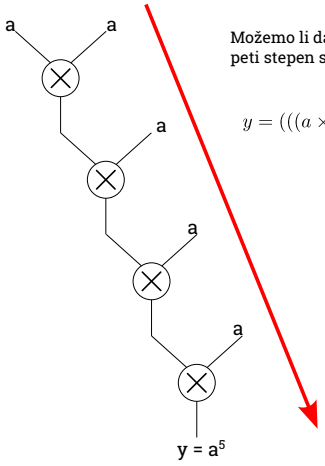

Nakon četiri kašnjenja množača,
na izlazu će se pojaviti rezultat

# Stepenovanje množačima




Možemo li da izračunamo peti stepen sa tri kašnjenja?

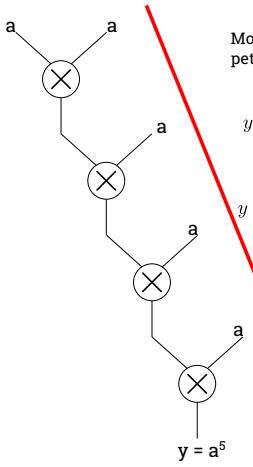
# Stepenovanje množačima




Možemo li da izračunamo
peti stepen sa tri kašnjenja?

$$y = (((a \times a) \times a) \times a) \times a$$

# Stepenovanje množačima

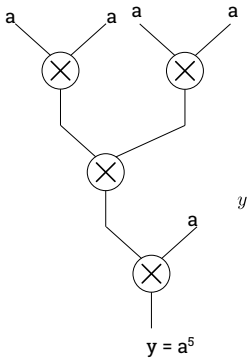



Možemo li da izračunamo
peti stepen sa tri kašnjenja?

$$y = (((a \times a) \times a) \times a) \times a$$

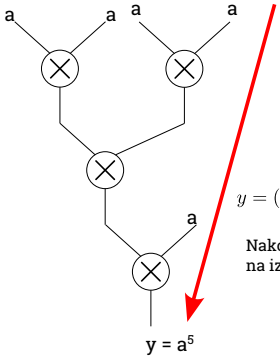
$$y = ((a \times a) \times (a \times a)) \times a$$

# Stepenovanje množačima




$$y = ((a \times a) \times (a \times a)) \times a$$

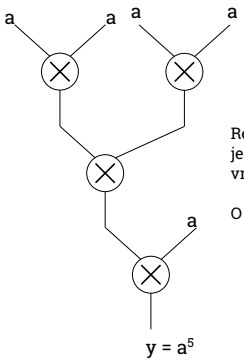
# Stepenovanje množačima




$$y = ((a \times a) \times (a \times a)) \times a$$

Nakon tri kašnjenja množača,
na izlazu će se pojaviti rezultat

# Stepenovanje množačima



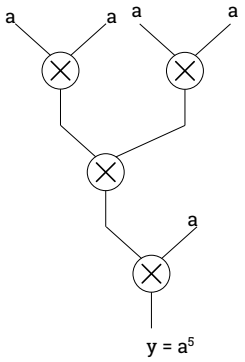

Redukcija visine stabla računanja je jedna od optimizacija koda koje vrše optimizujući kompajleri

O tome više uskoro

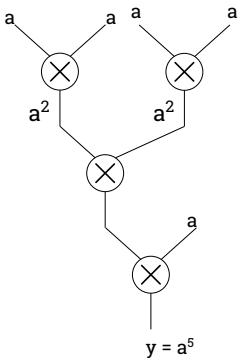
# Stepenovanje množačima

Da li su nam dovoljna tri množača?

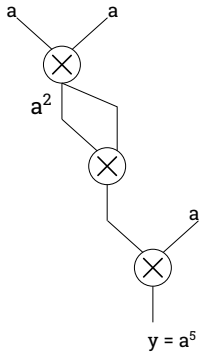
# Stepenovanje množačima



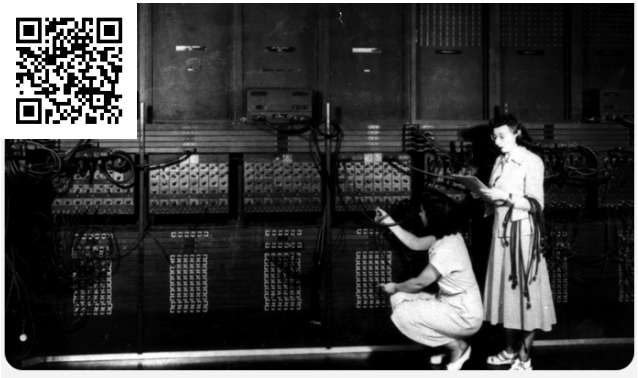
# Stepenovanje množačima



# Stepenovanje množačima



# ENIAC



Ruth Lichterman and Marlyn Wescoff reprogramming ENIAC in 1946

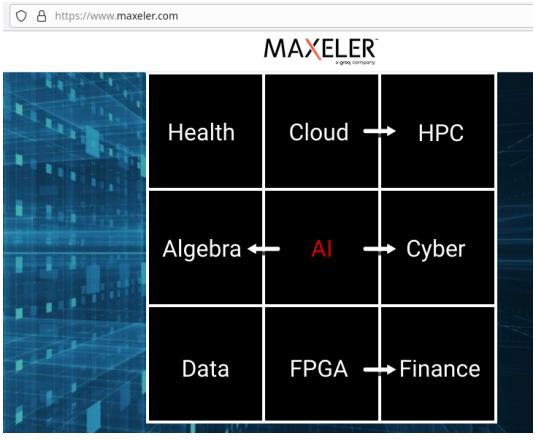
ENIAC je bio programiran povezivanjem funkcionalnih jedinica pomoću žica kako bi se oformilo kolo koje evaluira željenu funkciju.
Inače, svi prvi programeri su bili žene.

# High Frequency Trading Acceleration using FPGAs

*Christian Leber, Benjamin Geib, Heiner Litz*
University of Heidelberg
68131 Mannheim, Germany
{christian.leber, benjamin.geib, heiner.litz}@ziti.uni-heidelberg.de

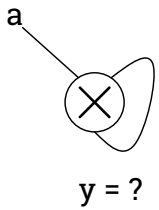
Povezivanje funkcionalnih jedinica vezama na takozvanim FPGA čipovima i do danas ostaje osnova prostornog računarstva (eng. *Spatial Computing*) koje, zbog vrlo malog kašnjenja, ima značajnu primenu u finansijama

# Prostorni računari



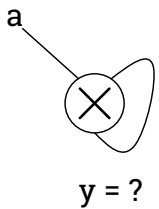
Možemo li da izračunamo peti stepen koristeći samo jedan množač?

# Vremenski multipleks




Da li će se na izlazu u nekom trenutku pojaviti $a^5$?

# Vremenski multipleks




Šta nam nedostaje da bismo to mogli da garantujemo?

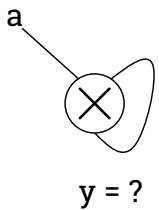
# Vremenski multipleks


Moramo da obezbedimo poznato početno stanje ulaza množača
(na primer, $a^0 = 1$)

# Bitna napomena

U računarstvu najčešće nije moguće pretpostaviti da će neki signal/promenljiva imati neku početnu vrednost, već ju je potrebno eksplicitno inicijalizovati!

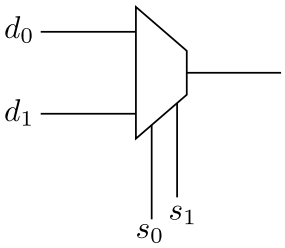
# Vremenski multipleks




Dakle, potrebno je da na početku na desni ulaz množača dovedemo $a^0 = 1$, a da mu zatim prosledimo izlaz množača

# Vreme je da uvedemo još jednu komponentu: multiplekser

Koristićemo $=_2$ da označimo izraze u $\mathbb{Z}_2$

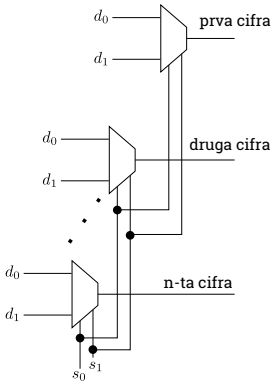



$y =_2 d_0 s_0 + d_1 s_1$

Najjednostavnije, one-hot kodovanje selekcionih ulaza

$s_0 + s_1 = 1$

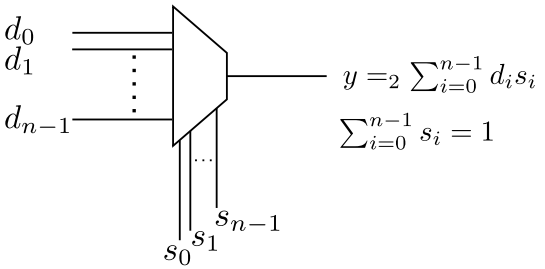
# Multipleksiranje višecifrenih binarnih brojeva




Naravno, možemo multipleksirati i višecifrene binarne brojeve: za svaku binarnu cifru dodamo još jedan multiplekser koji deli selekcione ulaze sa svim ostalim pojedinačnim multiplekserima

# Multipleksiranje više ulaznih signala

Kao i, na primer, vektorske prostore, i multipleksere možemo generalizovati na više ulaza, pri čemu se u svakom trenutku tačno jedan ulaz preslikava na izlaz



Srećom, zahvaljujući matematičarima znamo kako da u hardveru implementiramo bilo koju Bulovu funkciju

# Claude Shannon

66 languages

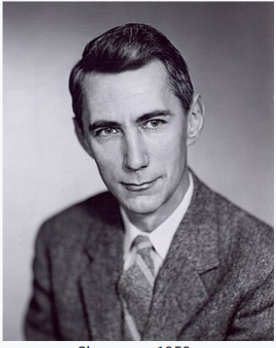
Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

**Claude Elwood Shannon** (April 30, 1916 – February 24, 2001) was an American mathematician, electrical engineer, computer scientist and cryptographer known as the "father of information theory". He was the first to describe the boolean gates (electronic circuits) that are essential to all digital electronic circuits, and he built the first machine learning device, thus founding the field of artificial intelligence.[1][2][3][4] He is credited alongside George Boole for laying the foundations of the Information Age.[5][6][7][4]

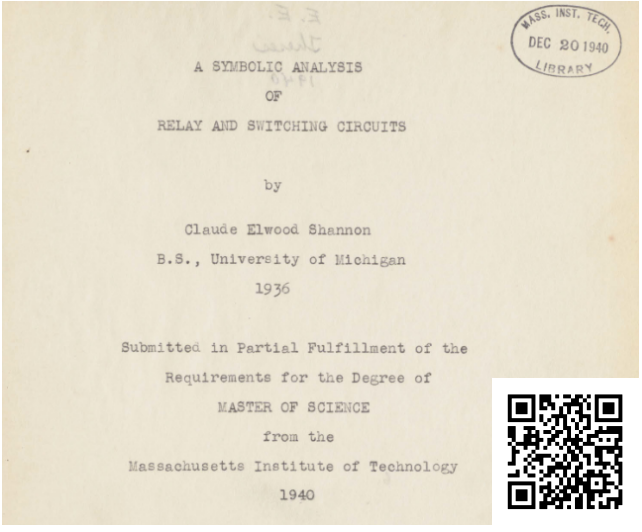
As a 21-year-old master's degree student at the Massachusetts Institute of Technology (MIT), he wrote his thesis demonstrating that electrical applications of Boolean algebra could construct any logical numerical relationship,[8] thereby establishing the theory behind digital computing and digital circuits.[9][10] In 1987, Howard Gardner called his thesis "possibly the most important, and also the most famous, master's thesis of the century",[11] and Herman Goldstine described it as "surely ... one of the most important master's theses ever written ... It helped to change digital circuit

**Claude Shannon**



Shannon c. 1950s

Srećom, zahvaljujući matematičarima znamo kako da u hardveru implementiramo bilo koju Bulovu funkciju

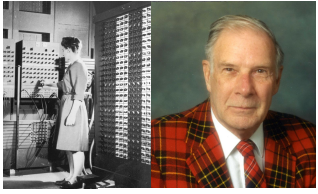


A SYMBOLIC ANALYSIS
OF
RELAY AND SWITCHING CIRCUITS

by

Claude Elwood Shannon
B.S., University of Michigan
1936

Submitted in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
from the
Massachusetts Institute of Technology
1940



**Richard Hamming**

**``You and Your Research''**

Transcription of the
Bell Communications Research Colloquium Seminar
7 March 1986

When I came to Bell Labs, I came into a very productive department. Bode was the department head at the time; Shannon was there, and there were other people. I continued examining the questions, ``Why?'' and

АЛГЕБРА ДВУХПОЛЮСНЫХ СХЕМ, ПОСТРОЕННЫХ ИСКЛЮЧИТЕЛЬНО ИЗ ДВУХПОЛЮСНИКОВ (АЛГЕБРА А-СХЕМ)[1]

*Канд. физ.-мат. наук* **В. И. Шестаков**

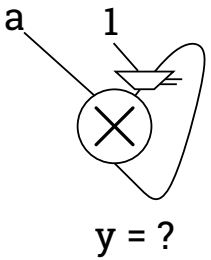
ВЗАИМНО-ОДНОЗНАЧНОЕ СООТВЕТСТВИЕ МЕЖДУ А-СХЕМАМИ И А-ВЫРАЖЕНИЯМИ

А-схемы

Viktor Ivanovič Šestakov, 1935.
(u pripremi doktorata na Lomonosovu)

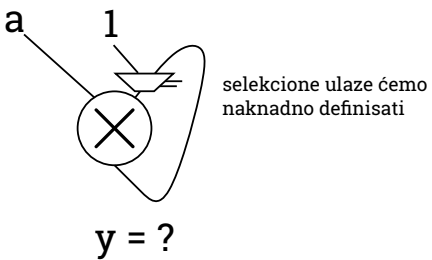
| | |
|---|---|
| **Дата рождения** | 15 октября 1907 |
| **Место рождения** | Москва, Российская империя |
| **Дата смерти** | 3 мая 1987 (79 лет) |
| **Место смерти** | Москва, СССР |
| **Страна** | Российская империя<br>СССР |
| **Научная сфера** | математика |
| **Место работы** | ИТМиВТ, МГУ |
| **Альма-матер** | МГУ |
| **Учёное звание** | профессор |
| **Научный руководитель** | Гливенко, Валерий Иванович |

# Da li će se sada na izlazu množača u nekom trenutku pojaviti $a^5$?



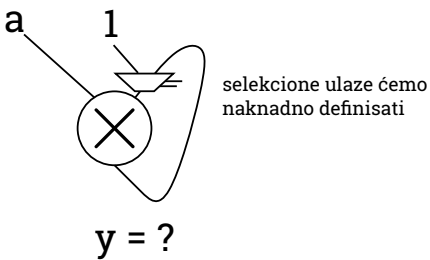

selekcione ulaze ćemo naknadno definisati

# Da li će se sada na izlazu množača u nekom trenutku pojaviti $a^5$?



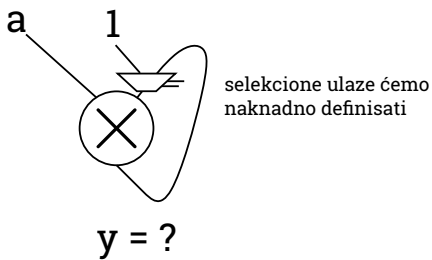

Hoće, ali će ga ubrzo zameniti $a^6$, pa $a^7$...

# Da li će se sada na izlazu množača u nekom trenutku pojaviti $a^5$?



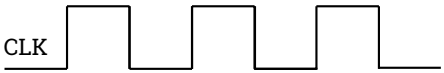

Šta nam onda još nedostaje?

# Da li će se sada na izlazu množača u nekom trenutku pojaviti $a^5$?



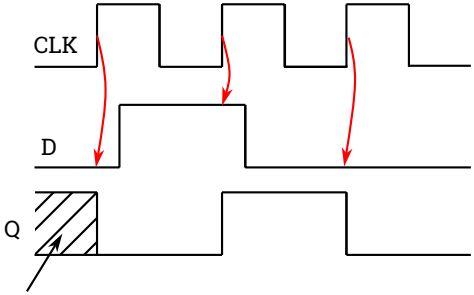

Moramo nekako da izbrojimo 5 množenja i onda zaustavimo dalje množenje

# Vreme je da uvedemo još jednu komponentu: registar

# Vreme je da uvedemo još jednu komponentu: registar

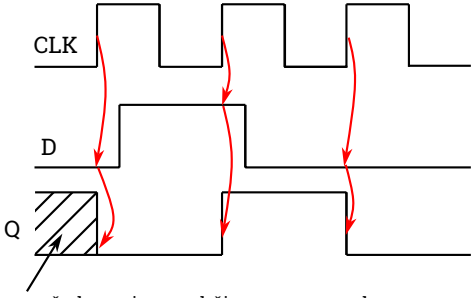

Kada naiđe rastuća ivica takta,
registar proverava vrednost ulaza D




na početku registar sadrži nepoznatu vrednost
(zato inicijalizujemo promenljive)

# Vreme je da uvedemo još jednu komponentu: registar

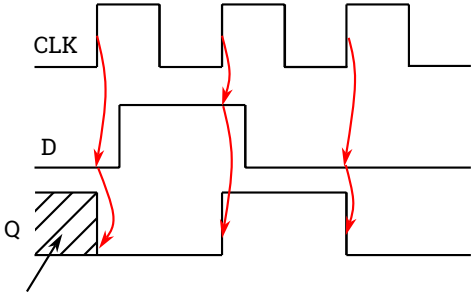

Kada naiđe rastuća ivica takta,
registar proverava vrednost ulaza D
i preslikava je na izlaz Q



# Vreme je da uvedemo još jednu komponentu: registar


Kada naiđe rastuća ivica takta,
registar proverava vrednost ulaza D
i preslikava je na izlaz Q

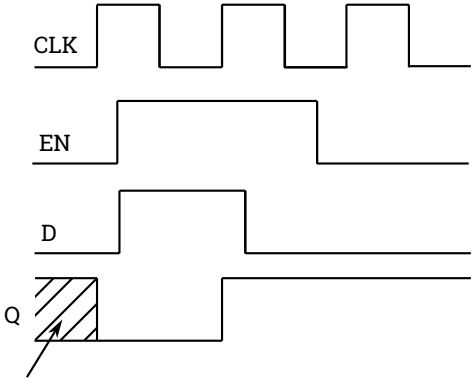
U međuvremenu, Q zadržava prethodnu vrednost




na početku registar sadrži nepoznatu vrednost
(zato inicijalizujemo promenljive)

# Vreme je da uvedemo još jednu komponentu: registar

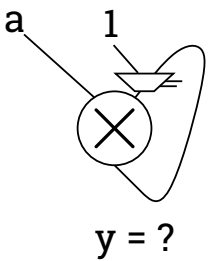

EN je signal za dozvolu upisa
Kada je 1, registar normalno funkcioniše
Kada je 0, Q ne menja vrednost, bez obzira na takt



# Ukratko o registrima

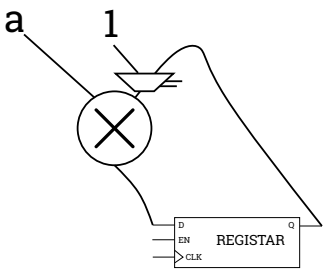
- Registar je osnovni višebitni memorijski element
- Signal takta diskretizuje vreme
- Registar sprečava „protrčavanje” vrednosti
- Signal EN omogućuje čuvanje vrednosti i kada naiđe rastuća ivica takta

# Gde je potrebno da ubacimo registar da bismo sprečili protrčavanje viših stepena?

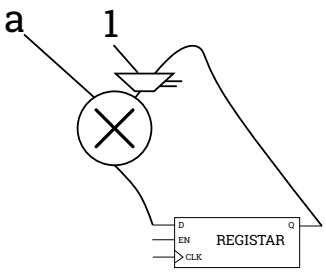



selekcione ulaze ćemo naknadno definisati

# Gde je potrebno da ubacimo registar da bismo sprečili protrčavanje viših stepena?

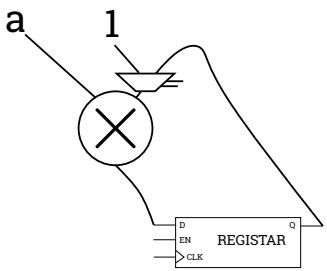


# Gde je potrebno da ubacimo registar da bismo sprečili protrčavanje viših stepena?



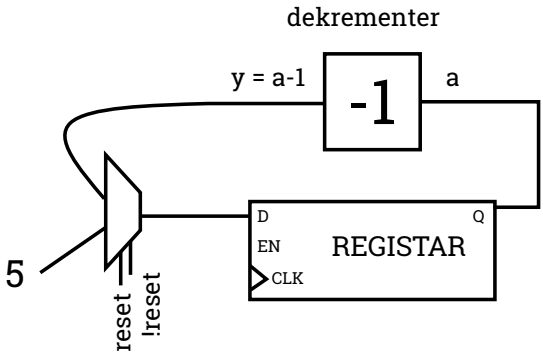

Šta nam još fali?

# Gde je potrebno da ubacimo registar da bismo sprečili protrčavanje viših stepena?

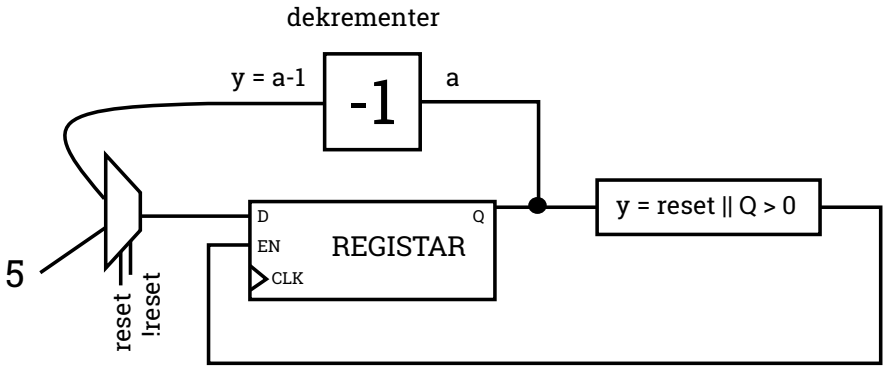



Brojač množenja, da bismo ispravno selektovali ulaze množača i da bismo zaustavili množenje na vreme

# Brojač već znamo da napravimo

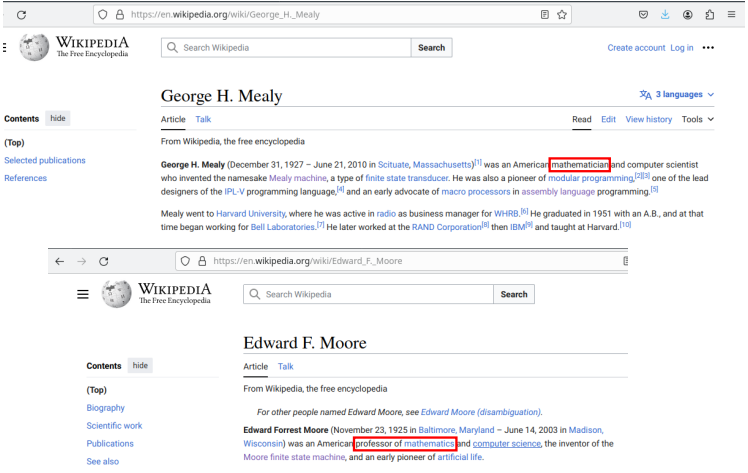


# Brojač već znamo da napravimo

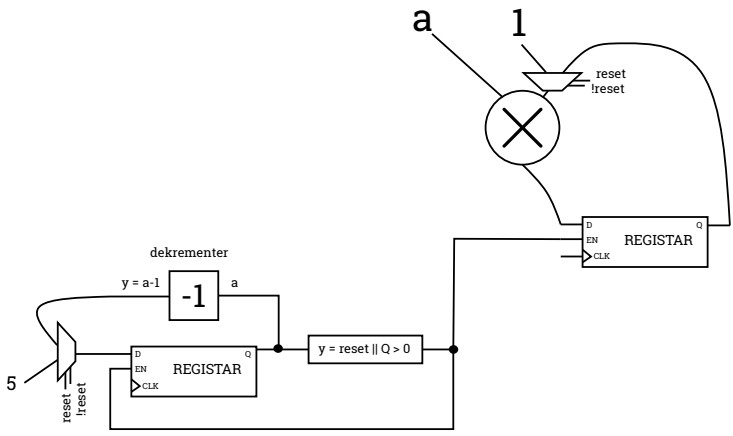


# Brojač

Brojač je specijalan slučaj konačnog automata (eng. Finite State Machine). Konačne automate su također izmislili matematičari i danas predstavljaju fundamentalni model za sintezu hardvera, ali se često koriste i u programiranju.



# Kako izgleda celo kolo za stepenovanje jednim množačem?



# Šta smo dobili ovim?

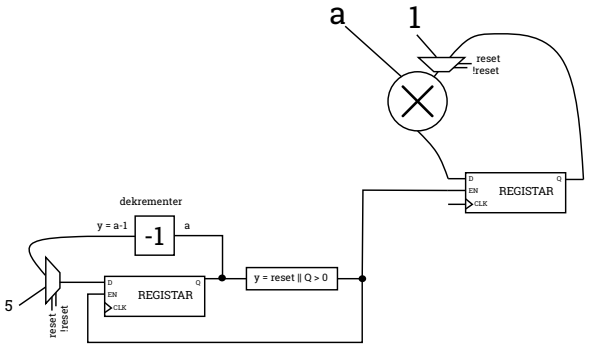
- Umesto tri, koristimo samo jedan množač

U vreme nastanka ENIAC-a, aritmetički sklopovi poput množača su bili izuzetno skupi za realizaciju, pa je smanjenje njihovog broja bilo od velikog značaja

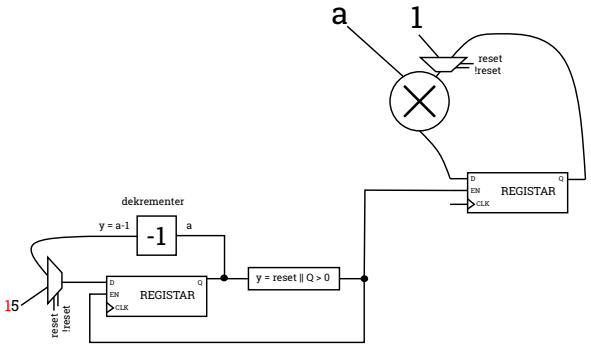
# Šta smo još dobili?

Kako bi izgledalo kolo za računanje 15. stepena?

# Šta smo još dobili?



# Šta smo još dobili?




Više ne moramo da ponovo povezujemo komponente kako bismo izračunali neki drugi stepen. Potrebno je samo da neku drugu vrednost upišemo u odgovarajući registar. To je ključni koncept „vremenskog računanja" (eng. *Temporal Computing*) koje je danas dominantno. U vreme ENIAC-a, povezivanje je iziskivalo ručni rad, pa je promena paradigme i iz tog razloga bila veoma važna. Na današnjim FPGA čipovima, dešava se automatski.

# Uopštavanje

# Kako da računamo nešto složenije od stepena?

Uzmimo za primer izraz oblika $\sum_{i=0}^{N-1} a_i^{n_i}$ i jednu konkretnu instancu $a_0^3 + a_1^2 + a_2^4 + a_3$

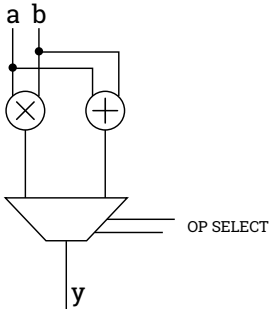
# Koje su nam operacije potrebne?

- sabiranje
- množenje

# Šta smo do sad naučili?

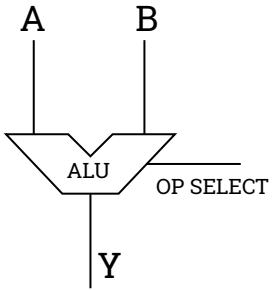
- Možemo da implementiramo bilo koju Bulovu funkciju
- To uključuje i multiplekser kao specijalan slučaj

# ALU

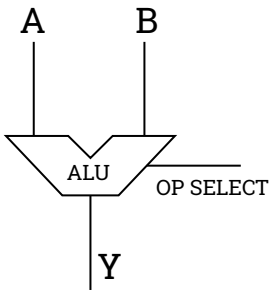


# ALU

Naravno, možemo uključiti i više od dve operacije. U opštem slučaju, ovakav blok nazivamo „aritmetičko-logičkom jedinicom” (eng. *Arithmetic Logic Unit* (ALU))

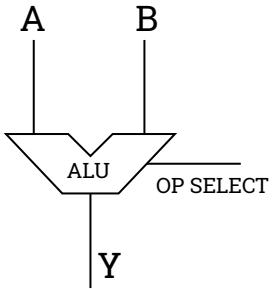


# ALU



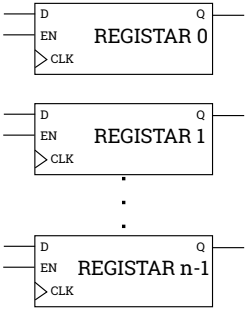

Da bismo izračunali $a_0^3 + a_1^2 + a_2^4 + a_3$, šta je potrebno da menjamo u vremenu?

# ALU

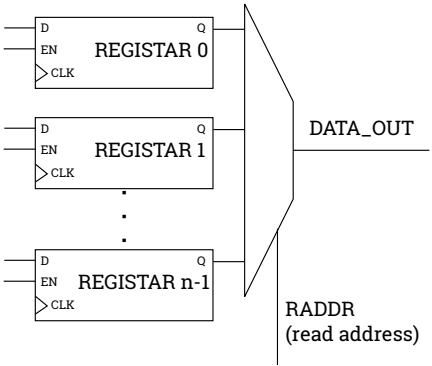



Potrebno je da menjamo operaciju i operande

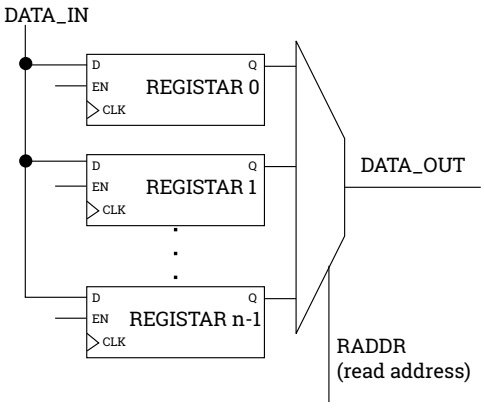
# Ponovo slažemo kockice: registarska banka



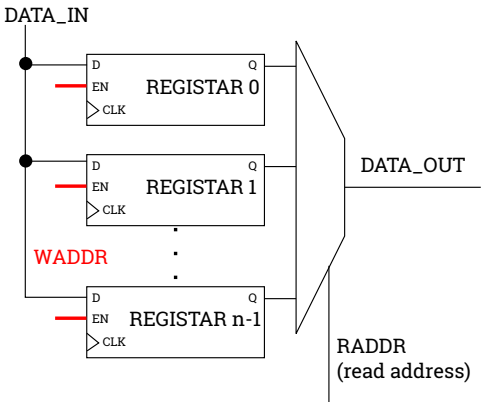
# Ponovo slažemo kockice: registarska banka



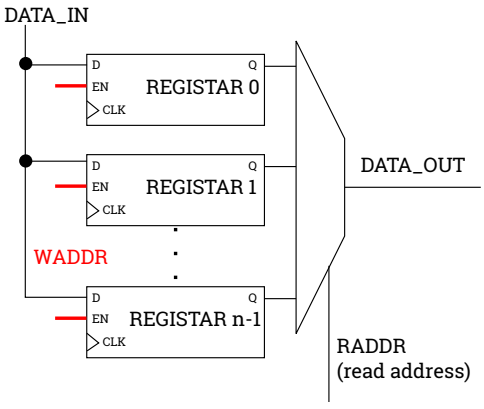
# Ponovo slažemo kockice: registarska banka



# Ponovo slažemo kockice: registarska banka



# Ponovo slažemo kockice: registarska banka




Par (RADDR, DATA_OUT) nazivamo „pristupom za čitanje” (eng. *Read Port*), a par (WADDR, DATA_IN) „pristupom za upis” (eng. *Write Port*)

# Analogija sa bankom

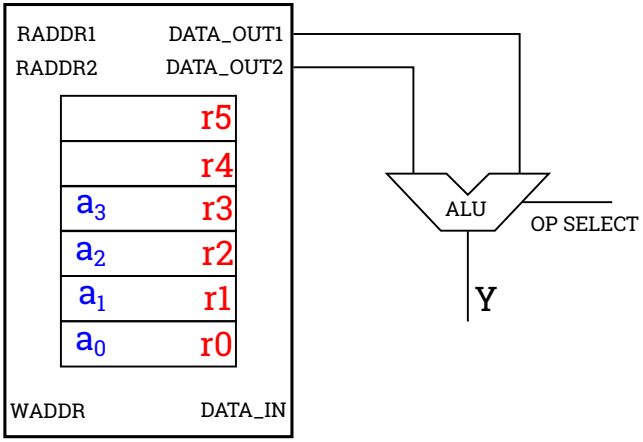
Jedan registar u registarskoj banci = jedan sef u trezoru banke

Jedan pristup registarske banke = jedan šalter u banci

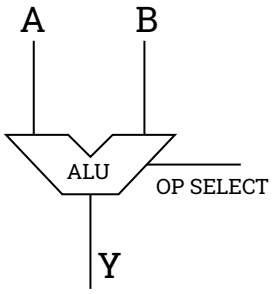
Sefova je mnogo više nego šaltera (registara je mnogo više nego pristupa)

Za razliku od deponovanja i povlačenja predmeta iz sefa, upis u registar briše prethodni sadržaj, dok ga čitanje ne uklanja

# Šta je sada potrebno da menjamo tokom računanja?



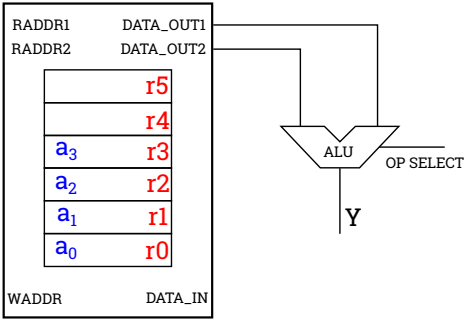
# ALU




Potrebno je da menjamo operaciju i ~~operande~~ adrese registara u kojima su smešteni operandi

# Koraci izmena

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



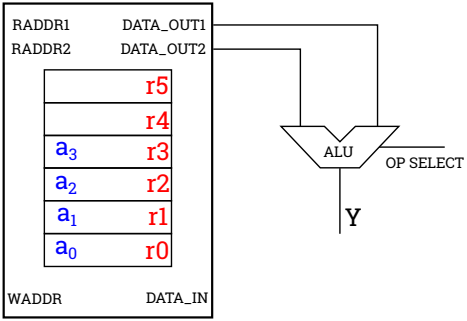

podatak upisan u registar

simbolička adresa registra

| Korak | RADDR1 | RADDR2 | OP |
|---|---|---|---|
| 1) | | | |

# Koraci izmena

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



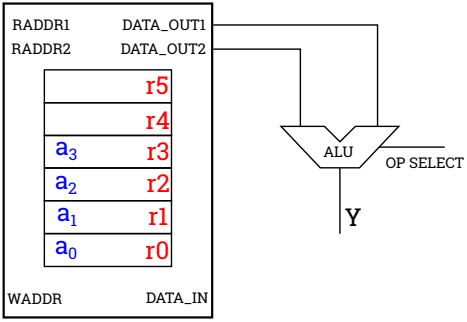

podatak upisan u registar

simbolička adresa registra

| Korak | RADDR1 | RADDR2 | OP |
|---|---|---|---|
| 1) | r0 | | |

# Koraci izmena

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



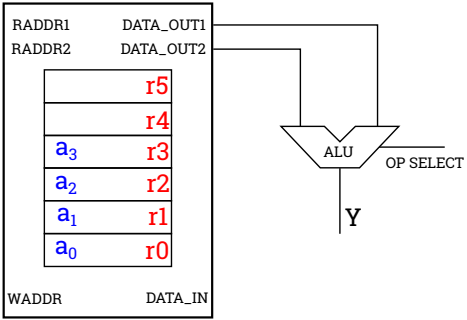

podatak upisan u registar

simbolička adresa registra

| Korak | RADDR1 | RADDR2 | OP |
|---|---|---|---|
| 1) | r0 | r0 | |

# Koraci izmena

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



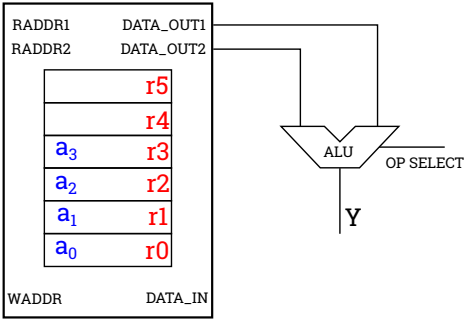

podatak upisan u registar

simbolička adresa registra

| Korak | RADDR1 | RADDR2 | OP |
|---|---|---|---|
| 1) | r0 | r0 | ✕ |

# Koraci izmena

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




podatak upisan u registar

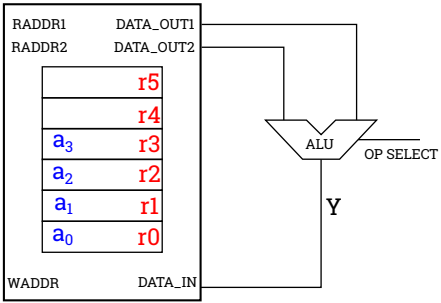
simbolička adresa registra

| Korak | RADDR1 | RADDR2 | OP |
|---|---|---|---|
| 1) | r0 | r0 | ✕ |

Šta radimo sa rezultatom?

# Koraci izmena

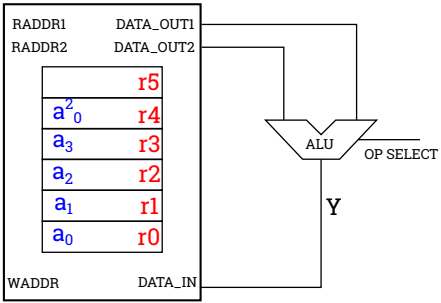
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | |

# Koraci izmena

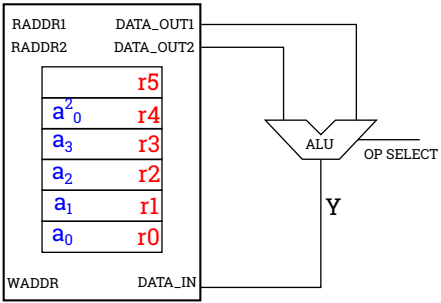
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | r4 |

# Koraci izmena

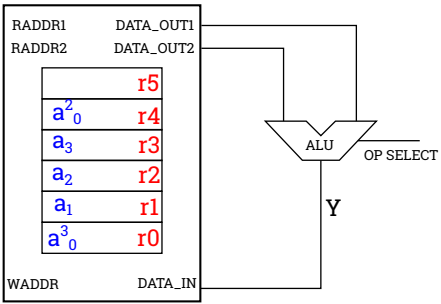
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | X | r4 |
| 2) | r4 | r0 | X | r0 |

# Koraci izmena

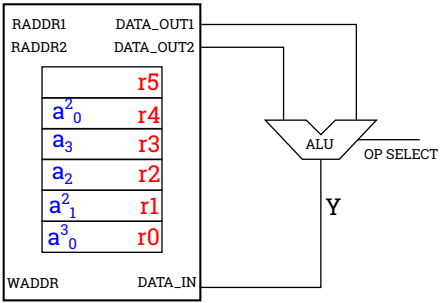
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | X | r4 |
| 2) | r4 | r0 | X | r0 |

# Koraci izmena

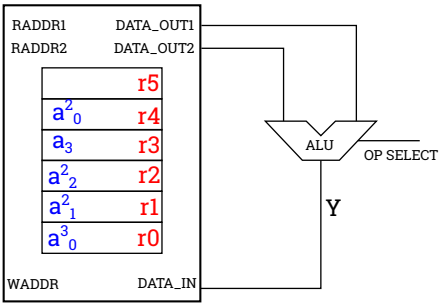
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | X | r4 |
| 2) | r4 | r0 | X | r0 |
| 3) | r1 | r1 | X | r1 |

# Koraci izmena

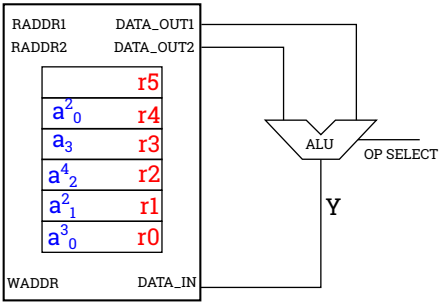
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | r4 |
| 2) | r4 | r0 | × | r0 |
| 3) | r1 | r1 | × | r1 |
| 4) | r2 | r2 | × | r2 |

# Koraci izmena

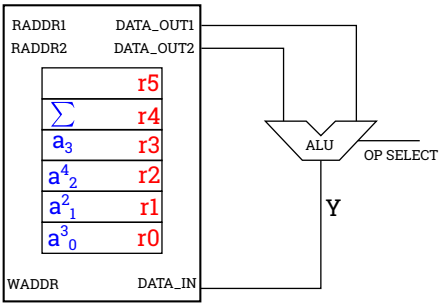
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | r4 |
| 2) | r4 | r0 | × | r0 |
| 3) | r1 | r1 | × | r1 |
| 4) | r2 | r2 | × | r2 |
| 5) | r2 | r2 | × | r2 |

# Koraci izmena

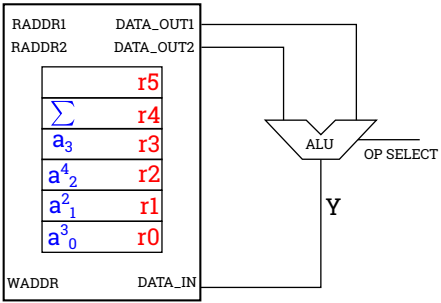
$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$




| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | r4 |
| 2) | r4 | r0 | × | r0 |
| 3) | r1 | r1 | × | r1 |
| 4) | r2 | r2 | × | r2 |
| 5) | r2 | r2 | × | r2 |
| 6) | r0 | r1 | + | r4 |

# Koraci izmena

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



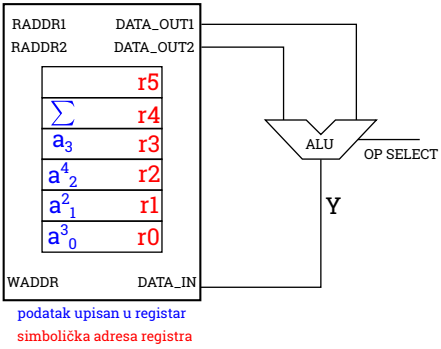

| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | r4 |
| 2) | r4 | r0 | × | r0 |
| 3) | r1 | r1 | × | r1 |
| 4) | r2 | r2 | × | r2 |
| 5) | r2 | r2 | × | r2 |
| 6) | r0 | r1 | + | r4 |
| 7) | r4 | r2 | + | r4 |
| 8) | r4 | r3 | + | r4 |

# Kako da omogućimo računaru da sam izvrši te korake?

# Kako da omogućimo računaru da sam izvrši te korake?

Svaka instrukcija je zapravo broj (setite se multipleksera)

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



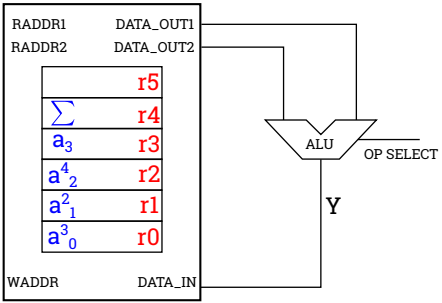

| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | r4 |
| | 000001 | 000001 | 10 | 010000 |

0000010000011001000

# Kako da omogućimo računaru da sam izvrši te korake?

Svaka instrukcija je zapravo broj (setite se multipleksera)

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



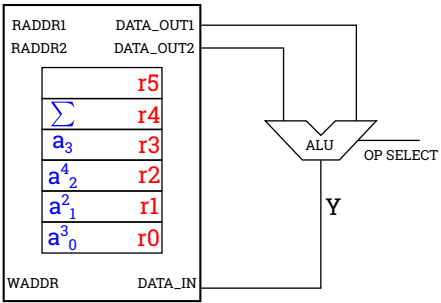

| Korak | RADDR1 | RADDR2 | OP | WADDR |
|---|---|---|---|---|
| 1) | r0 | r0 | × | r4 |
| | 000001 | 000001 | 10 | 010000 |

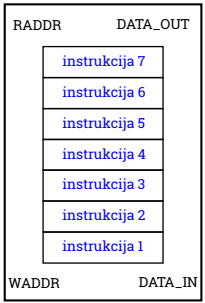
0000010000011001000 0

Taj niz brojeva možemo da smestimo u posebnu memoriju

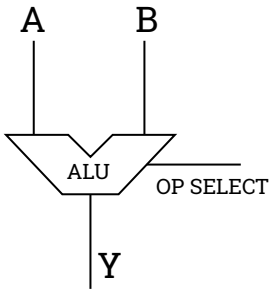
# Šta je sada potrebno da menjamo tokom računanja?

$$a_0^3 + a_1^2 + a_2^4 + a^3 = a_0 \times a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 \times a_2 \times a_2 + a_3$$



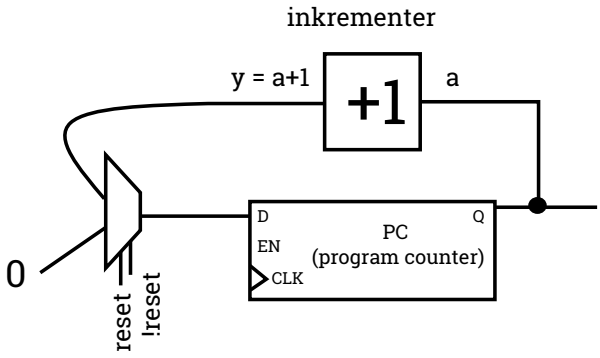
# ALU



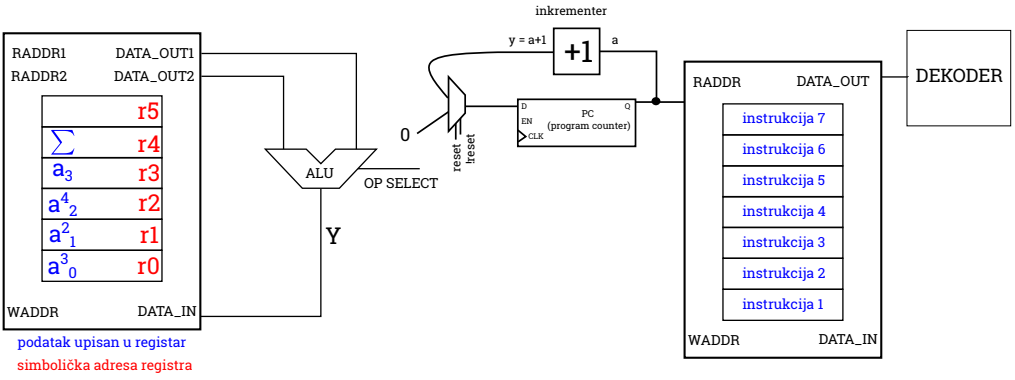

Potrebno je da menjamo ~~operaciju i operande adrese registara u kojima su smešteni operandi~~ adresu instrukcije u memoriji

Kako menjamo adresu?

# Ponovo nam je potreban brojač



# Ponovo nam je potreban brojač



Ovaj tip računara zovemo „računar sa sačuvanim programom” (eng. *Stored-Program Computer*) jer računar sam čita iz memorije niz instrukcija koje čine program koji je potrebno da izvrši

Taj tip arhitekture računara koji je danas apsolutno dominantan nazivamo još i „Fon Nojmanova arhitektura”

Izmislili su je Ekert i Mokli, prilikom dizajna naslednika ENIAC-a, pod imenom EDVAC

Formalni opis koncepata uvedenih u EVAC, prvi je dao Džon fon Nojman (eng. John von Neumann), pa po njemu arhitektura sve do danas nosi ime

# Džon fon Nojman

von Neumann in the 1940s

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

# Džon fon Nojman

von Neumann in the 1940s

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

Šta mislite, šta je po struci bio fon Nojman?

# Džon fon Nojman

von Neumann in the 1940s

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

Tako je, matematičar

# Džon fon Nojman

von Neumann in the 1940s

First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W–670–ORD–4926

Between the

United States Army Ordnance Department

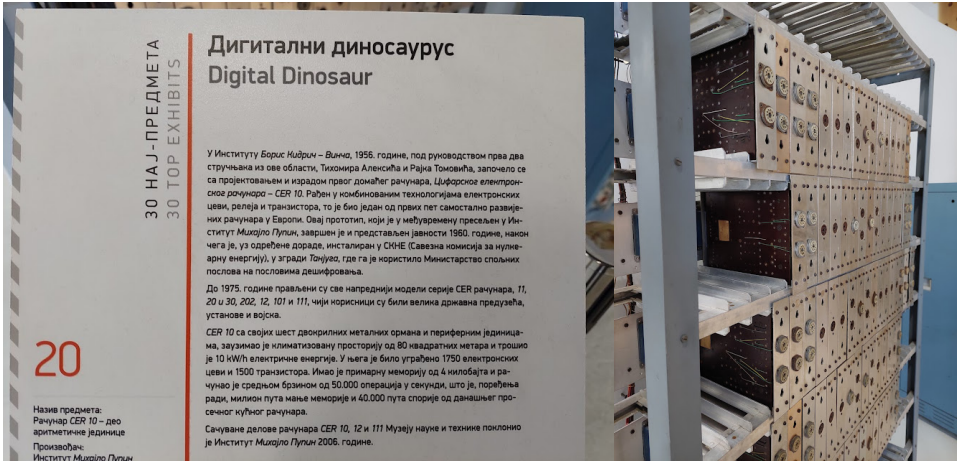
and the

University of Pennsylvania

A šta mislite, gde je rođen i gde je završio gimnaziju?


Universitätstrasse 6
6 Universitätstrasse
Zürich, Zurich
Google Street View
Jul 2021
Terra – ETH Zürich
Google
Image capture: Jul 2021 © 2024 Google Serbia Terms Privacy Report a problem

# Inače, Jugoslavija je bila peta zemlja u Evropi koja je imala sopstveni digitalni računar



30 НАЈ-ПРЕДМЕТА
30 TOP EXHIBITS

## Дигитални диносаурус
## Digital Dinosaur

У Институту *Борис Кидрич – Винча*, 1956. године, под руководством прва два стручњака из ове области, Тихомира Аленсића и Рајка Томовића, започело се са пројектовањем и израдом првог домаћег рачунара, *Цифарског електронског рачунара – CER 10*. Рађен у комбинованим технологијама електронских цеви, релеја и транзистора, то је био један од првих пет самостално развијених рачунара у Европи. Овај прототип, који је у међувремену пресељен у Институт *Михајло Пупин*, завршен је и представљен јавности 1960. године, након чега је, уз одређене дораде, инсталиран у СКНЕ (Савезна комисија за нуклеарну енергију), у згради *Танјуга*, где га је користило Министарство спољних послова на пословима дешифровања.

До 1975. године прављени су све напреднији модели серије CER рачунара, *11, 20 и 30, 202, 12, 101 и 111*, чији корисници су били велика државна предузећа, установе и војска.

*CER 10* са својих шест двокрилних металних ормана и периферним јединицама, заузимао је климатизовану просторију од 80 квадратних метара и трошио је 10 kW/h електричне енергије. У њега је било уграђено 1750 електронских цеви и 1500 транзистора. Имао је примарну меморију од 4 килобајта и рачунао је средњом брзином од 50.000 операција у секунди, што је, поређења ради, милион пута мање меморије и 40.000 пута спорије од данашњег просечног кућног рачунара.

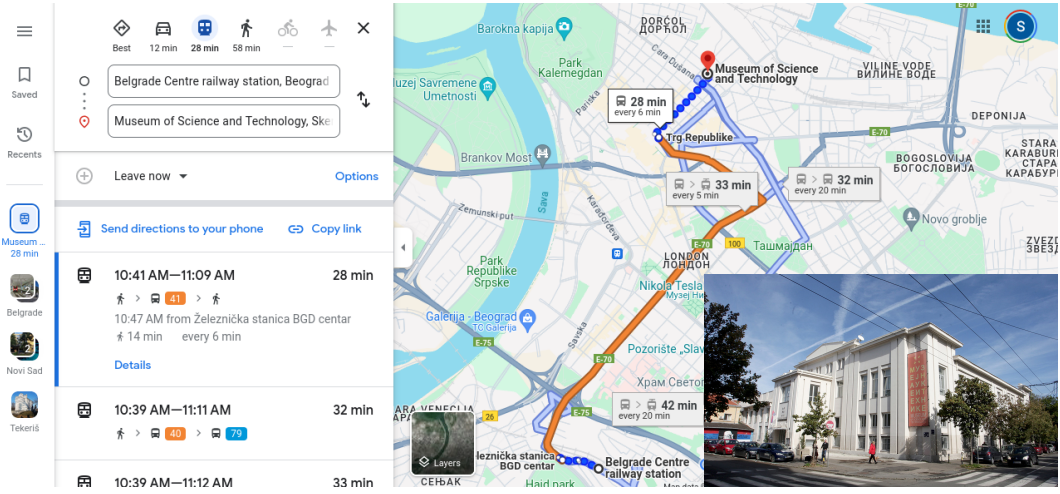
Сачуване делове рачунара *CER 10, 12* и *111* Музеју науке и технике поклонио је Институт *Михајло Пупин* 2006. године.

20

Назив предмета:
Рачунар *CER 10* – део аритметичке јединице

Произвођач:
Институт *Михајло Пупин*

# Inače, Jugoslavija je bila peta zemlja u Evropi koja je imala sopstveni digitalni računar

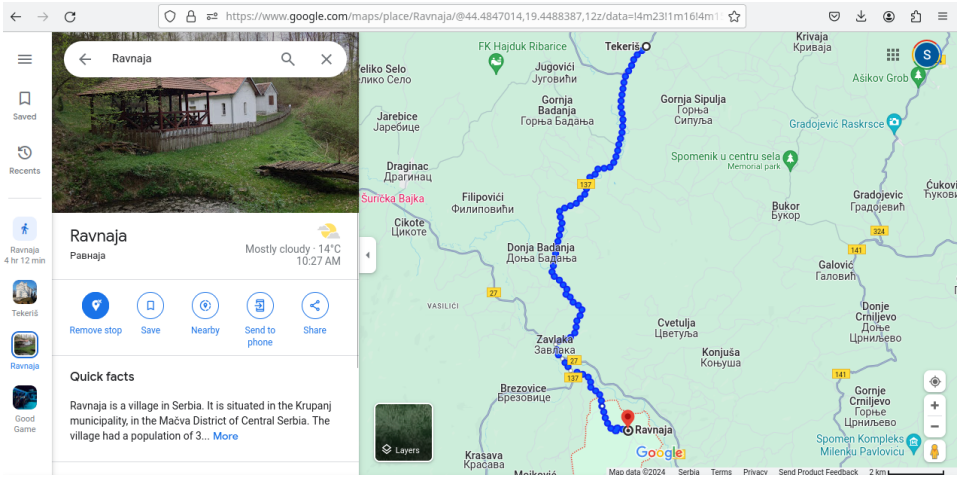


# Inače, Jugoslavija je bila peta zemlja u Evropi koja je imala sopstveni digitalni računar

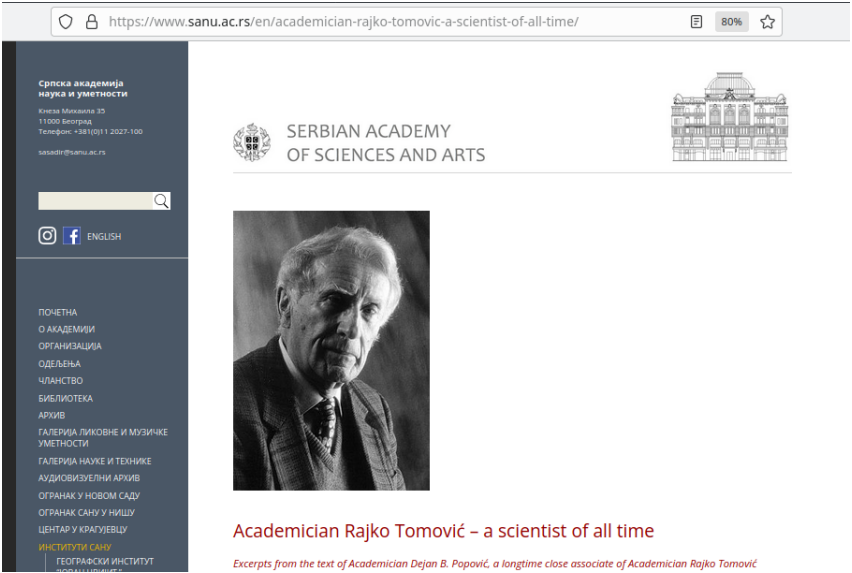
Tihomir Aleksić (Ravnaja, 1922-Beograd, 2004)

# Inače, Jugoslavija je bila peta zemlja u Evropi koja je imala sopstveni digitalni računar

CER = Cifarski elektronski računar



# Inače, Jugoslavija je bila peta zemlja u Evropi koja je imala sopstveni digitalni računar



# Nismo još završili

# Naš računar je još uvek veoma ograničen

Za sad smo videli instrukcije tipa

ADD r0, r1, r0

Program koji prevodi ovakve simboličke zapise u odgovarajuće binarne brojeve nazivamo „asemblerom”, a ovakav simbolički jezik „asemblijem” (eng. *assembly*). Za njihov nastanak je zaslužna prof. Ketlin But, koju smo pominjali na prošlom času

# Naš računar je još uvek veoma ograničen

Za sad smo videli instrukcije tipa

ADD r0, r1, r0

Prvi problem: oba operanda su specificirana navođenjem imena registra u kom se nalaze. Šta ako je neophodno da baratamo konstantama?

# Naš računar je još uvek veoma ograničen

Za sad smo videli instrukcije tipa

ADD r0, r1, r0

Nikakav problem. Uvodimo novu instrukciju tipa

ADDI r0, 0xN, r0

ADDI od eng. *add immediate*, što nalaže dekoderu da umesto da interpretira 0xN kao adresu, direktno prosledi taj broj ALU-u kao konstantan operand

# Modovi adresiranja

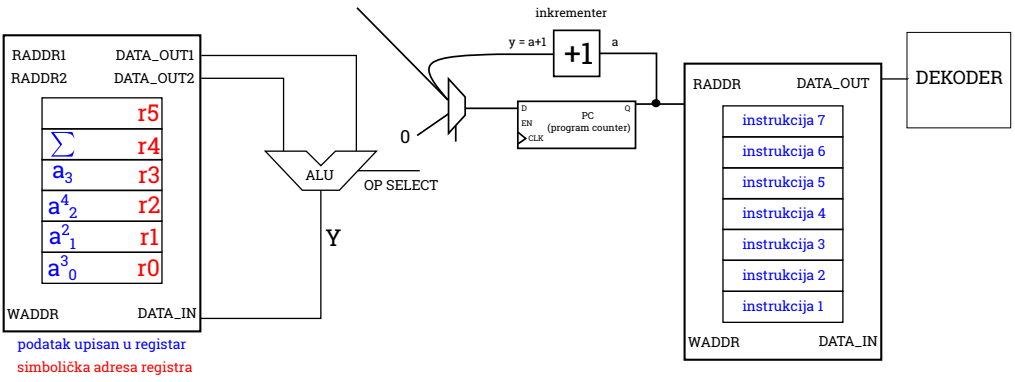
Razlika između instrukcija ADD i ADDI je u takozvanim „modovima adresiranja”. Ima ih još ali ćemo ih pomenuti tek kada budemo radili nizove u C/C++-u.

# Naš računar je još uvek veoma ograničen

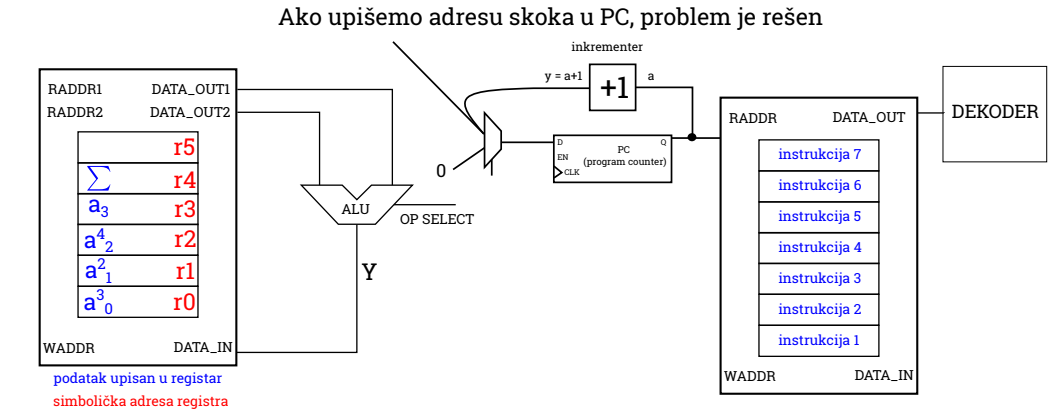
Daleko veći problem je to što može da izvršava samo linearne segmente koda (bez grananja)

No, i to vrlo lako možemo da rešimo

# Kako da skočimo na proizvoljnu instrukciju?



# Kako da skočimo na proizvoljnu instrukciju?




Gde specificiramo adresu skoka?

# Bezuslovni skok

Uvodimo novu instrukciju

JUMP TADDR

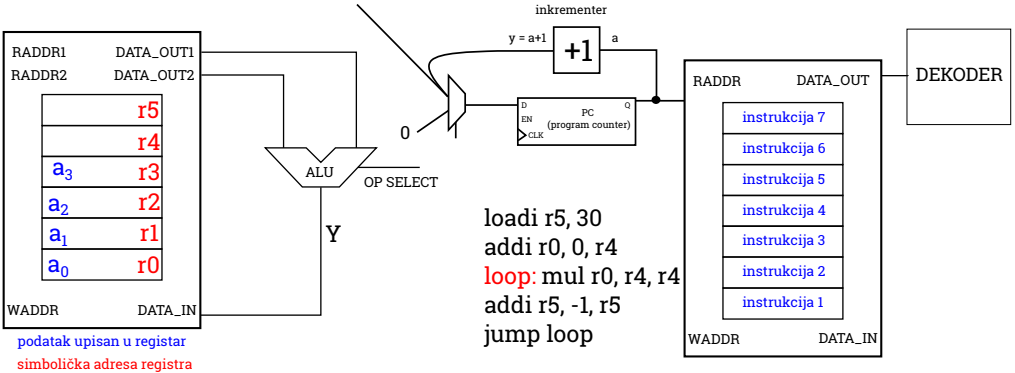
Kada se izvrši, vrednost u PC-u se menja vrednošću konstante TADDR

Kod svih drugih instrukcija koje smo do sada videli, sledeća instrukcija je implicitno na adresi PC + 1[1]

JUMP eksplicitno navodi adresu sledeće instrukcije

[1] Ili PC + 4, PC + 8, u zavisnosti od toga koliko bita ima procesor i da li je dužina svake instrukcije ista. To su za nas trenutno nebitni detalji, navedeni ovde samo radi kompletnosti.

# Šta radi ovaj kod?

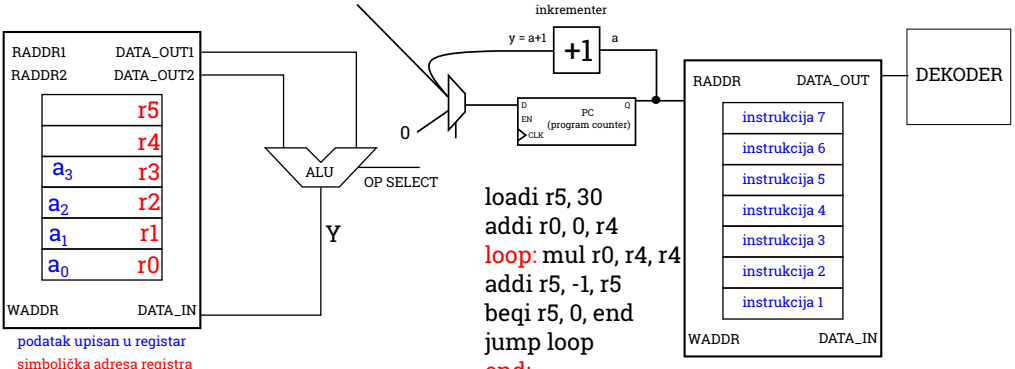


# Uslovni skokovi

Potrebne su nam i instrukcije skoka koji se izvršava samo ako je neki uslov zadovoljen.

Na primer, BEQI RA, 0xN, TADDR proverava da li registar RA sadrži broj 0xN i ako da, učitava TADDR u PC. U suprotnom, ne radi ništa.

# Šta radi ovaj kod?

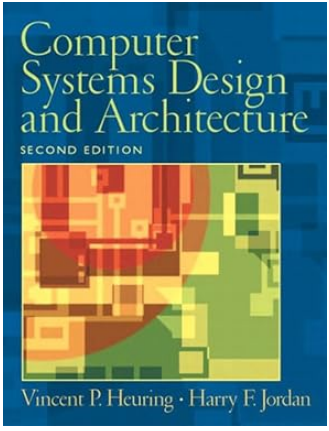


# Poziv funkcije je vrlo sličan

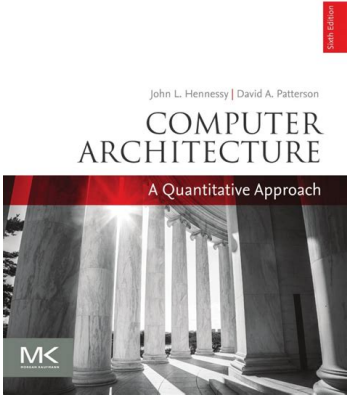
Potrebno je da zapamtimo u nekom rezervnom registru vrednost PC-a. Zatim vršimo bezuslovni skok na adresu na kojoj počinje telo pozvane funkcije. Kada stignemo do kraja, RET instrukcija vraća PC na staru vrednost.

Ima tu još detalja, ali vratićemo se na njih kada za to bude vreme.

# Ako neko želi da nauči više o arhitekturi računara




Odlična knjiga za uvod

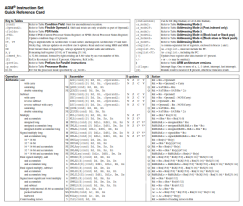



Nezvanična "Biblija"

# Šta radi kompajler?

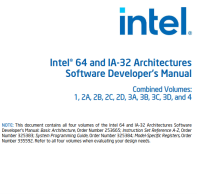
# Šta je glavni problem asemblija?

# Šta je glavni problem asemblija?


Svaka konkretna Arhitektura seta instrukcija (eng. *Instruction Set Arhcitecture* (ISA)) zahteva drugi kod

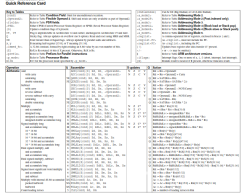
# Šta je glavni problem asemblija?


intel

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Combined Volumes:
1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4

NOTE: This document contains all four volumes of the Intel 64 and IA-32 Architectures Software Developer's Manual: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.


ARM® Instruction Set
Quick Reference Card



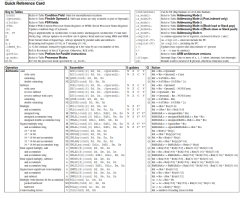
**The RISC-V Instruction Set Manual**
**Volume I: Unprivileged ISA**
Document Version 20191213


Editors: Andrew Waterman[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu

December 13, 2019

Program napisan za jedan procesor, ne bi radio na drugom

# Šta je glavni problem asemblija?


No, svaki računar zahteva neke podudarne vrste programa (na primer, operativni sistem)

# Ideja

Umesto da pišemo operativni sistem za svaki računar posebno, u odgovarajućem asembliju, napišemo ga u nekom višem jeziku, a proizvođač svakog računara (ili neko drugi) napiše program koji prevodi taj viši izvorni kod u odgovarajući skup instrukcija

# Ideja

To stvara ogromnu uštedu programerske radne snage, ali i obezbeđuje kvalitetniji softver (bolja optimizacija, otklanjanje grešaka...)

Programski jezik C je nastao upravo zbog potrebe da se u nekom višem programskom jeziku napiše operativni sistem Unix, kako bi bio portabilan.

Program koji prevodi izvorni kod napisan u višem programskom jeziku u niz instrukcija nazivamo „kompajlerom”

# Rekurzivna primena ideje

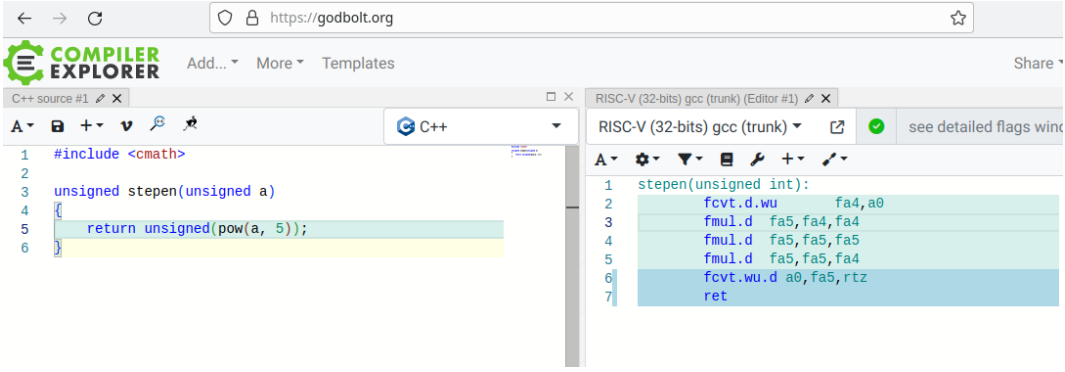
Pored pukog prevođenja, kompajler može i značajno da optimizuje kod. Većina raspoloživih optimizacija je opšta i ne zavisi od konkretnog procesora. Stoga se kod prvo prevodi u međureprezentaciju (eng. *intermediate representation* (IR)), koja se zatim optimizuje na različite načine (na primer, redukcijom visine stabla računanja, koju smo već videli, zatim uklanjanjem koda koji se nikada ne izvršava, itd). Ta optimizovana reprezentacija se zatim prevodi u niz instrukcija u fazi „generisanja koda" (eng. *code generation*). Samo ta poslednja faza zavisi od konkretnog računara za koji generišemo kod.

# Nešto o interpretiranim jezicima

Python je interpretiran jezik, što znači da program koji nazivamo interpreterom prevođenje vrši u letu, tokom izvršenja.

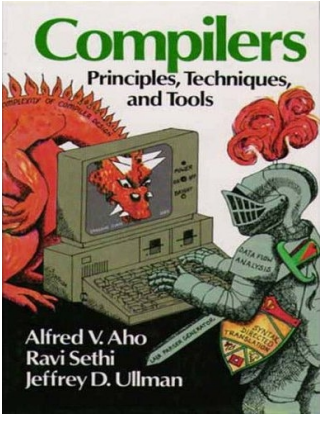
Posledice: 1) vreme provedeno u prevođenju ulazi u vreme izvršenja, 2) optimizacije je moguće vršiti samo nad ograničenim segmentima koda, jer interpreter ne sagledava ceo program odjednom. Upravo to je glavni razlog zašto su C/C++ programi i po nekoliko redova veličine puta brži.

# Kako kompajler implementira stepen sa prošlog časa?




Na ovom sajtu se možete igrati raznim kompajlerima i raznim ISA-ma, bez ikakve instalacije

# Ako neko želi da nauči više o kompajlerima



# Osnovna struktura C/C++ programa

# Zdravo svete

```cpp
//Nas prvi program. Ovo je komentar koji traje do kraja linije.
#include <iostream>
/*
 * Ovo je pretprocesorska direktiva koja nalaže pretprocesoru
 * ---programu koji se izvrsava neposredno pre kompilacije---
 * da u kod koji je potrebno kompajlirati uveze sadrzinu fajla
 * iostream u kom su, izmedju ostalih, navedena zaglavlja
 * funkcija za ulaz sa tastature i izlaz na ekran.
*/

int main()
/*
 * Definicija glavne funkcije. Nas program se ne izvrsava
 *  samostalno na procesoru, vec posredstvom operativnog sistema.
 *  Operativni sistem "skace" na pocetak naseg programa
 *   pozivom ove funkcije.
*/
{ //Pocetak bloka

    std::cout << "Zdravo svete!\n";
        //Svaka naredba u C/C++-u se zavrsava
        //takozvanim terminatorom>: znakom ;

        //std je takozvani imenski prostor (eng. namespace),
        //slican pozivnom broju drzave. Na primer, +38121 je Novi Sad
        //a +4121 Lozana.

        return 0; //Ako je program izvrsen bez gresaka, po konvenciji,
          //main vraca operativnom sistemu 0.

} //Kraj bloka
```

Hvala na pažnji