

Latvijas Universitāte
Datorikas fakultāte

CPU. DataPath

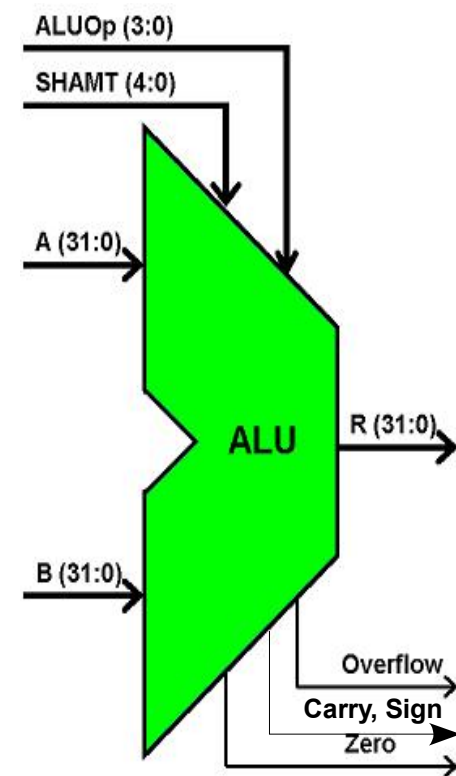
Kurss "ievads digitālajā projektēšanā"
Lekcija 05.10.2012

Autors: Rinalds Ruskuls

Aritmētiskais-loģiskais mezgls (**Arithmetic-Logic Unit**)

ALU pamatooperācijas

- Loģiskās (AND, OR, XOR, NOT)
- Aritmētiskās (“+”, “-”, “.”, “/”)
- Bitu nobīdes operācijas
- Salīdzināšanas operācijas.



MIPS

- **Microprocessor without Interclocked Pipeline Stages**
- Pieder pie **RISC** (**Reduced Instruction Set Computing**)
- Eksistē 32 un 64 bitu versijas (šajā kursā runāsim par 32 bitu versiju)
- Šāda tipa mikroprocesori tiek izmantoti rūteros (Cisco), spēļu konsolēs (PlayStation 1&2), kā arī citās iegultās sistēmās

Vientakts CPU implementācija

- Lekcijā tiks izstāstīta vienkārša *MIPS CPU* arhitektūras implementācija, kas atbalsta šādas operācijas:
 - Aritmētiskās
 - Datu ierakstīšana / nolasīšana
- Šodienas lekcijā tiks apskatīts vientakts MIPS CPU
 - Katra operācija izpildās vienā taktī
 - Uzsvars – datu maģistrāle

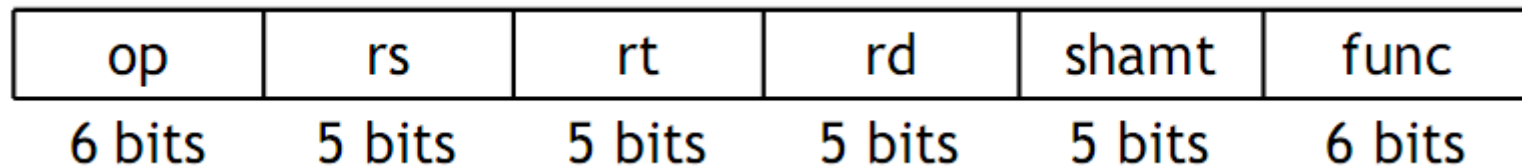
MIPS instrukciju tipi

Type	format (bits)					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

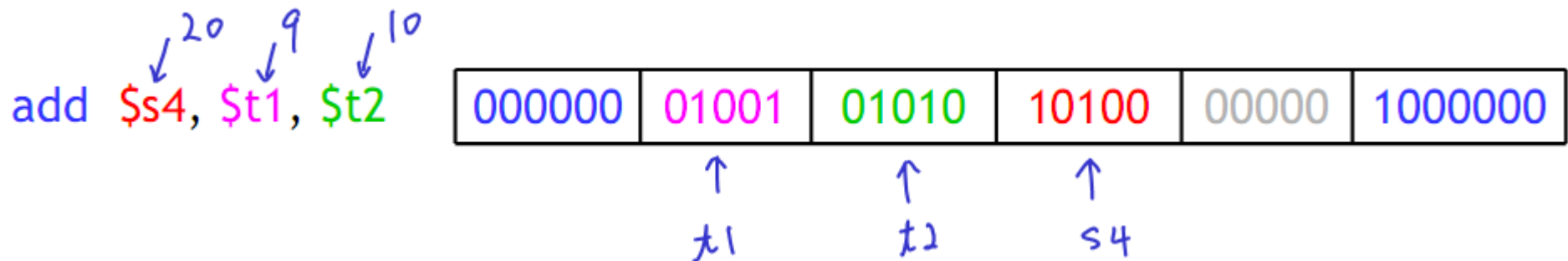
- Opcode – darbības tips
- rs,rt – ieejas reģistri, rd – izejas reģistrs
- Shamt – nobīdes vērtība
- FUNCT – darbības kods
- Immediate, adres – skaitliskās vērtības

R komandu izpildīšana

- Divu reģistru aritmētiskām darbībām izmanto R tipa instrukciju tipu
 - Op – instrukcijas opkods, func – nosaka izpildāmo aritmētisko operāciju
 - Rs, rt, rd ieejas, izejas reģistri



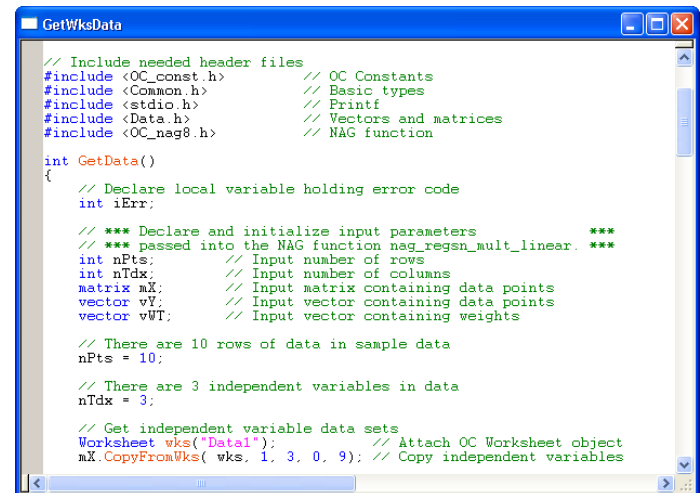
- Instrukcijas un tās atkodēšanas piemērs



Instrukciju atmiņa (Instruction Memory)

- Glabā programmu (instrukciju)
- Atgriež instrukciju pēc uzdotās adreses satura
- Vairums gadījumos nepieciešams tikai lasīšanas režīms:
 - DataPath neveic izmaiņas programmā
 - Programmas ievade atmiņā ir uzskatāma par specifiku gadījumu un var tikt realizēta atsevišķi

Ad	Instrukcija
A(0)	0000001111.....100000
A(1)	100000110010011..00
A(2)	10000111100000011..
A(3)	000000001110011..00
...	00000000111000...00
A(n)	000000001110011..00



```
GetWksData
// Include needed header files
#include <OC_const.h> // OC Constants
#include <Common.h> // Basic types
#include <stdio.h> // Printf
#include <Data.h> // Vectors and matrices
#include <OC_nag8.h> // NAG function

int GetData()
{
    // Declare local variable holding error code
    int iErr;

    // ** Declare and initialize input parameters **
    // ** passed into the NAG function nag_regsn_mult_linear. **
    int nPts; // Input number of rows
    int nTdx; // Input number of columns
    matrix mX; // Input matrix containing data points
    vector vY; // Input vector containing data points
    vector vWT; // Input vector containing weights

    // There are 10 rows of data in sample data
    nPts = 10;

    // There are 3 independent variables in data
    nTdx = 3;

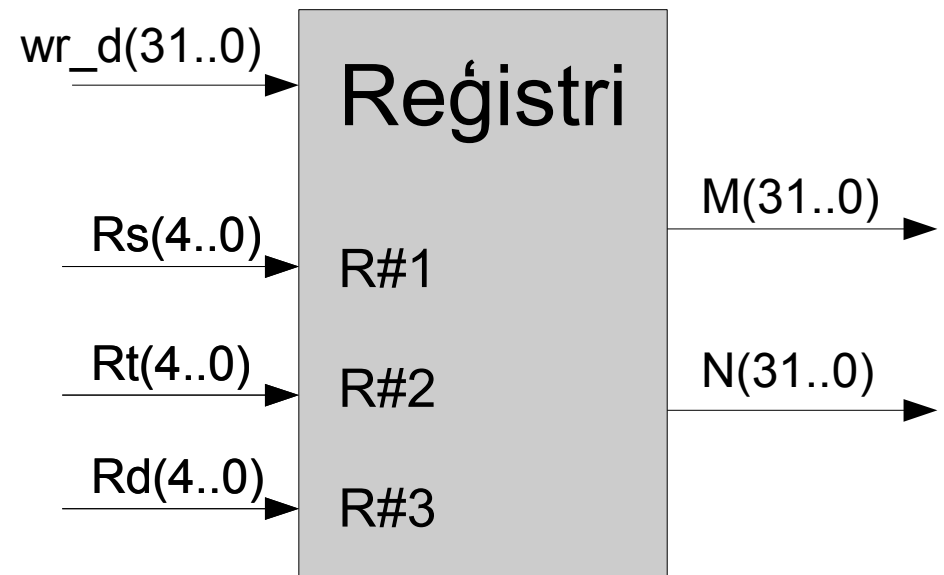
    // Get independent variable data sets
    Worksheet wks("Data1"); // Attach OC Worksheet object
    mX.CopyFromWks( wks, 1, 3, 0, 9); // Copy independent variables
```

Reģistru fails (Register file)

Reģistri – atmiņa, kurā iespējams rakstīt un lasīt

Lasīt ir iespējams divu reģistru vērtības vienlaicīgi – **M**, **N**

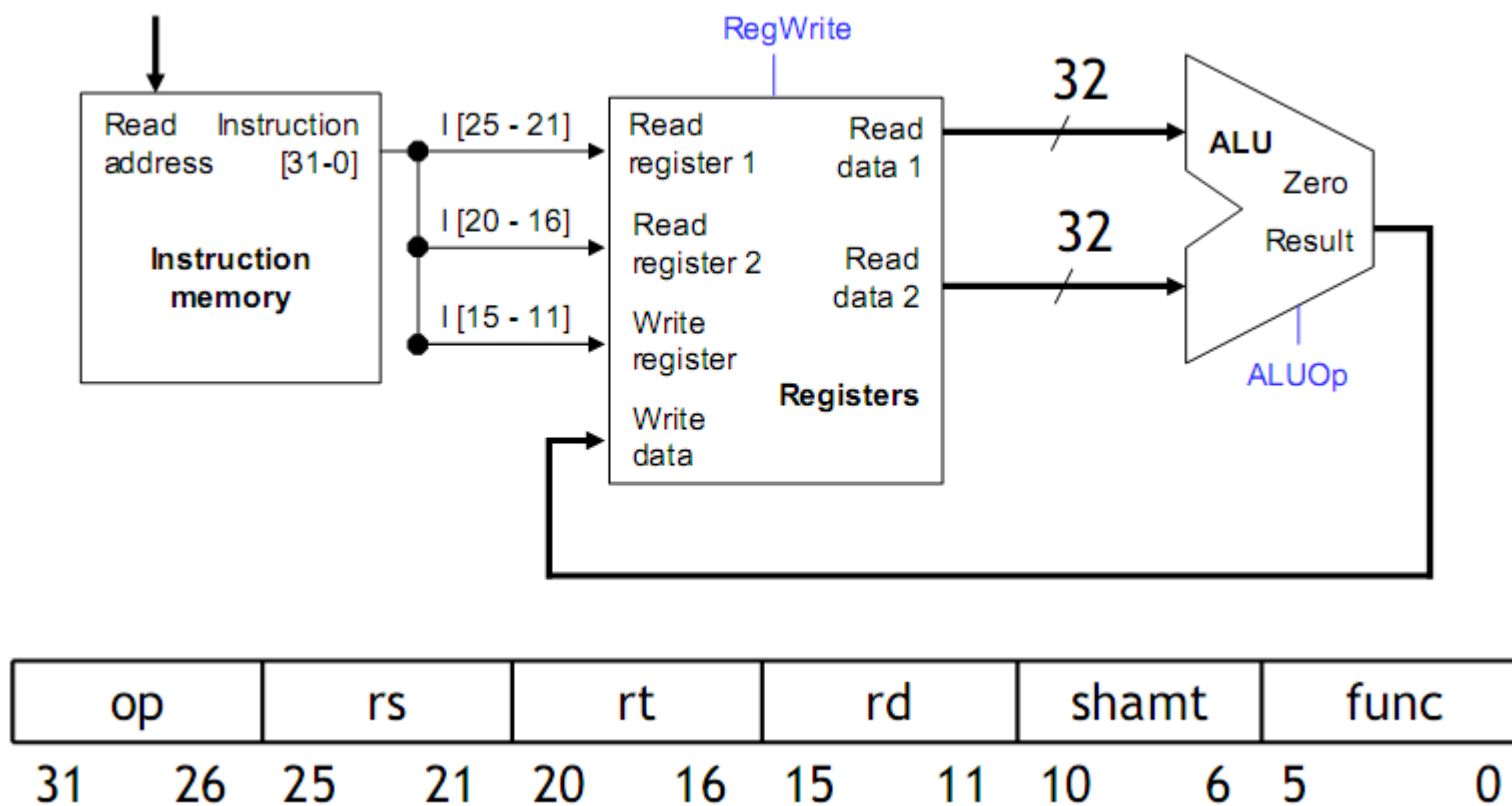
Rakstīt vienu vērtību - **wr_d**



Mūsu izmantotais reģistrs var saglabāt 32 - 32 bitu vērtības

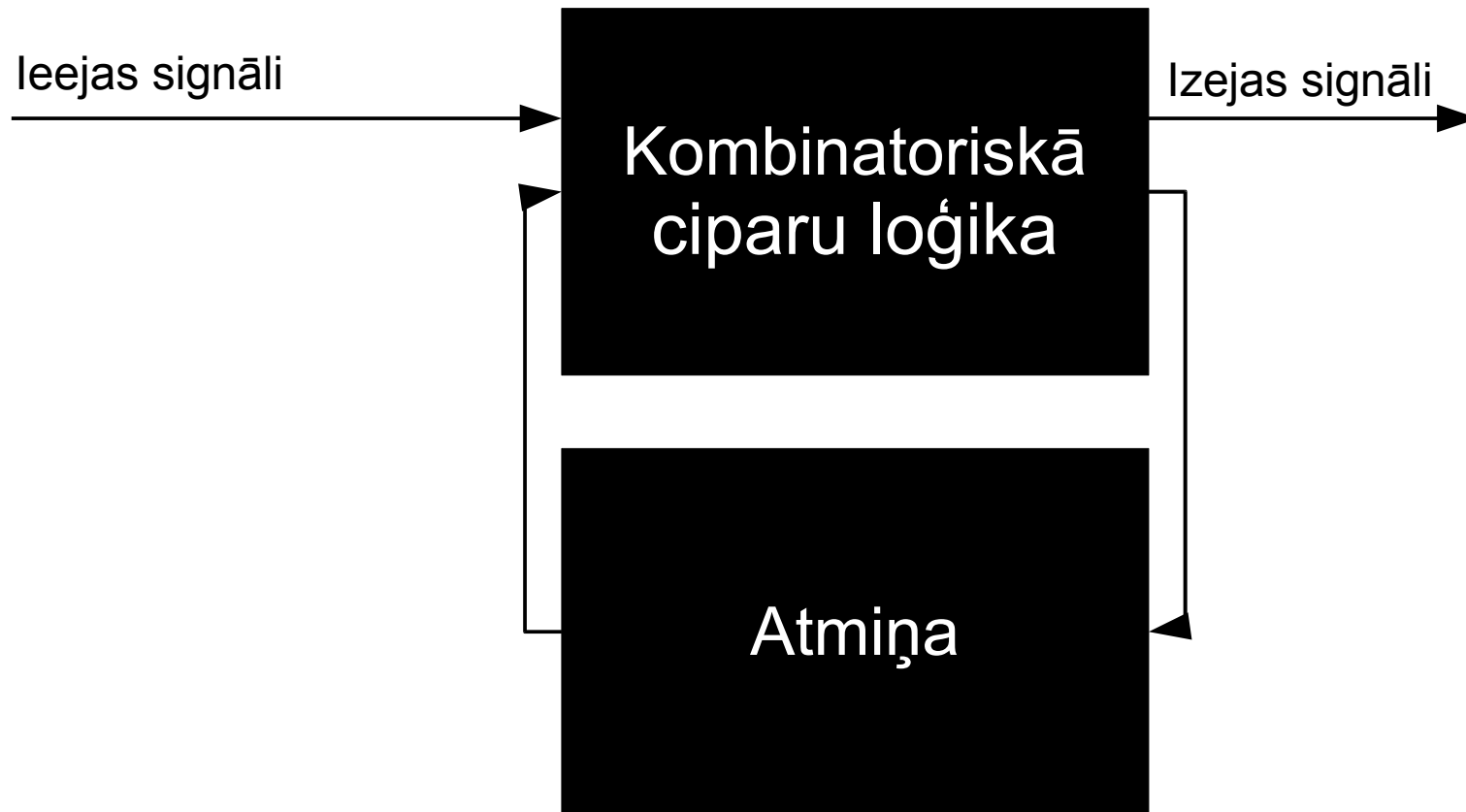
- ***rs***, ***rt***, ***rd*** – definē reģistra kārtas numuru

- 1) Nolasa instrukciju no instrukciju atmiņas – padodot adresi
- 2) Definētie reģistri (*rs*, *rt*) tiek nolasīti no reģistru faila
- 3) ALU – izpilda aritmētisko operāciju
- 4) Rezultāts tiek salgabāts izejas reģistrā *rd*



CPU. Datapath. ALU

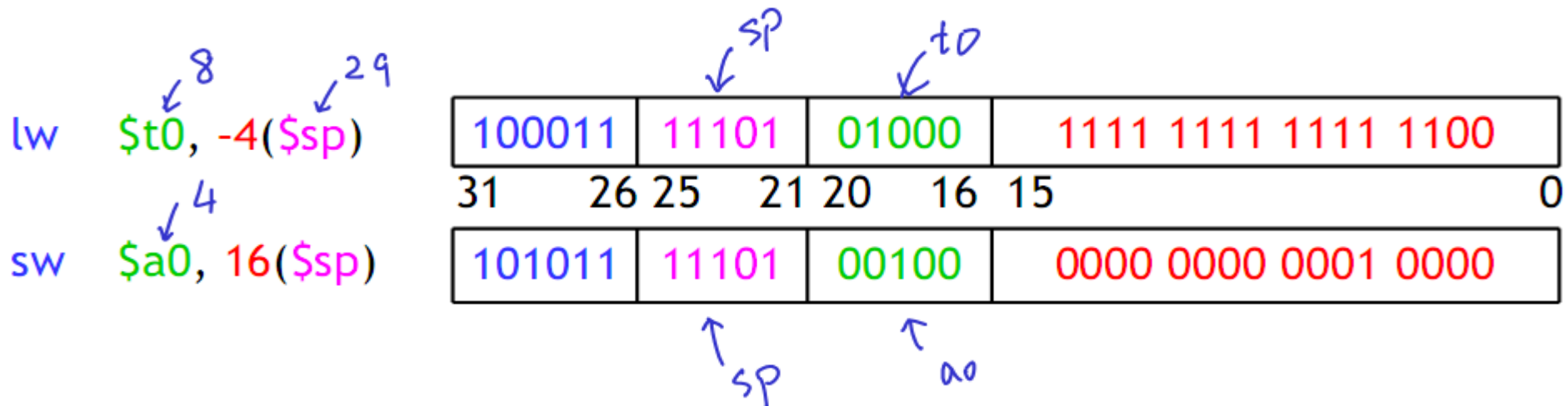
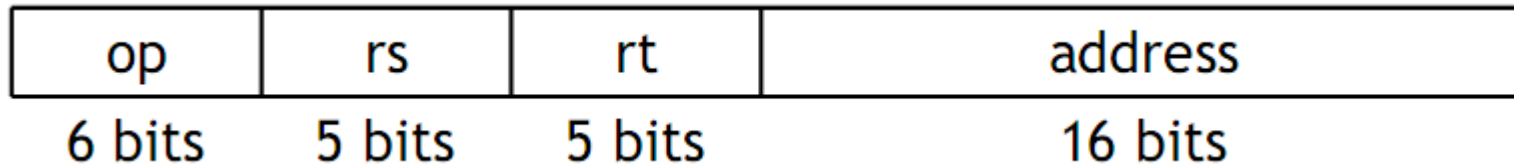
Ir līdzības ar iepriekšējo shēmu?



I komandu izpildīšana

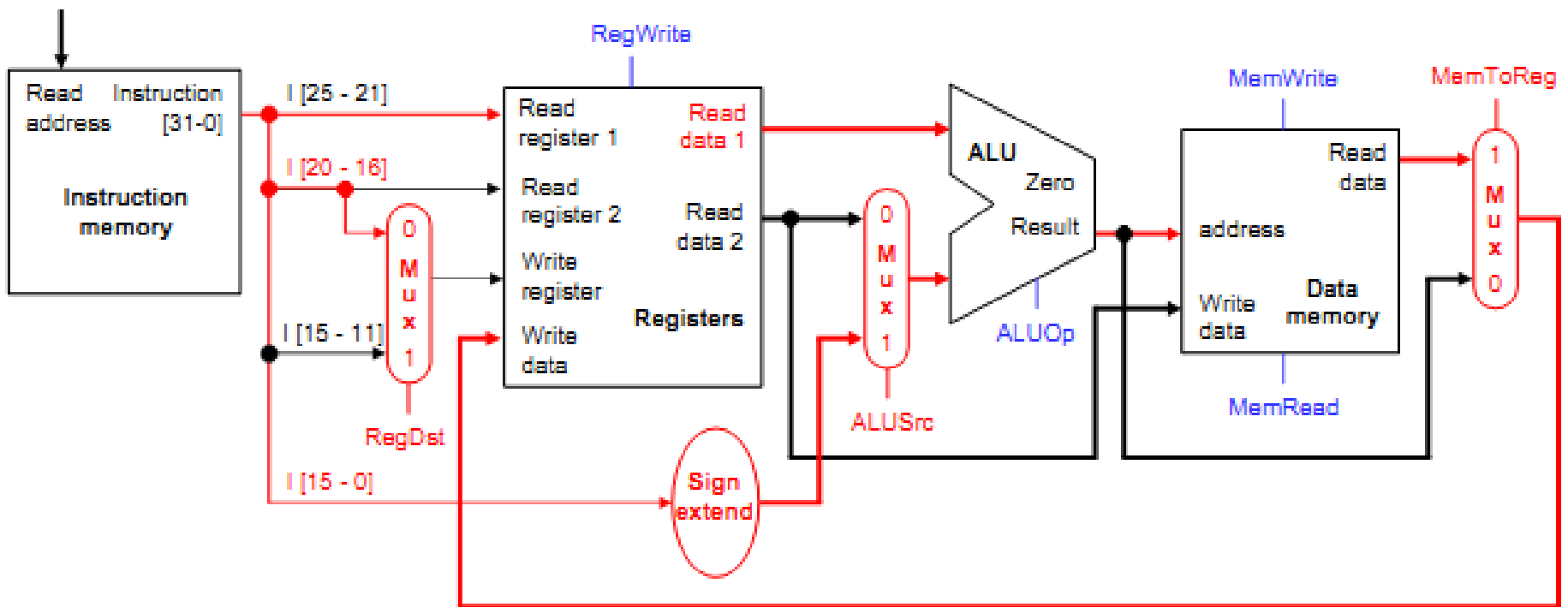
Komandas kā lw, sw, beq izmanto I tipa kodējumu

- ▶ rt ir izejas reģistrs priekš lw, bet priekš sw, beq ieejas
- ▶ address – 16 bitu konstante



Shēmas kopskats

Ja nepieciešams izpildīt šādu operāciju – `lw $t0, -4($sp)`, bāzes reģistrs `$sp` ir saskaitīts ar **sign-extend** konstanti, adreses vērtības iegūšanai. Nozīmē, ka ALU jāspēj saņemt gan reģistru vērtības, gan **sign-extend immediate** operandus priekš `lw` un `ws` komandām, levietojot multiplexoru – tiek panākta vēlāmā funkcionalitāte.



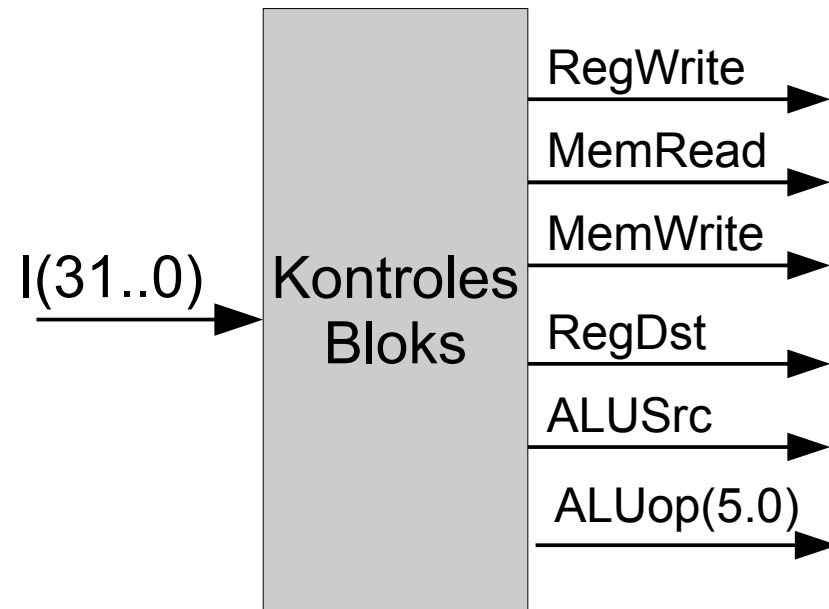
CPU. Datapath. ALU

Kontroles bloks

Shēmas daļa, kas rūpējas par to, lai katra operācija izpildītos pareizi

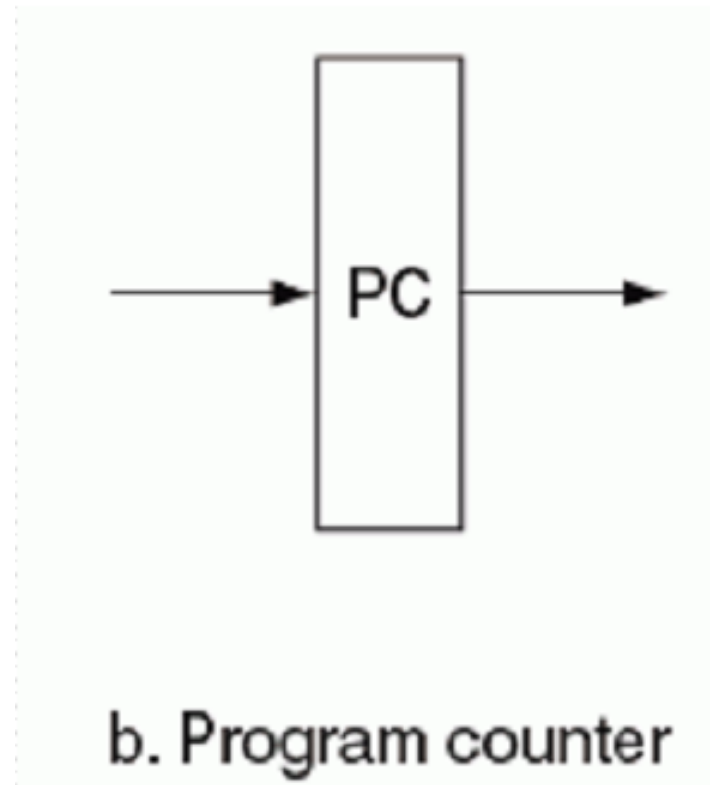
- › Ieejā ir instrukcija – bitu virkne (32 bitu gara)
- › Izejā signāli, kas kontrolē multipleksorus, ALU utt (iepriekšējās shēmās attēloti zilā un sarkanā krāsā)

Vairums signālu tiek ģenerēti izejot no opkoda vērtības



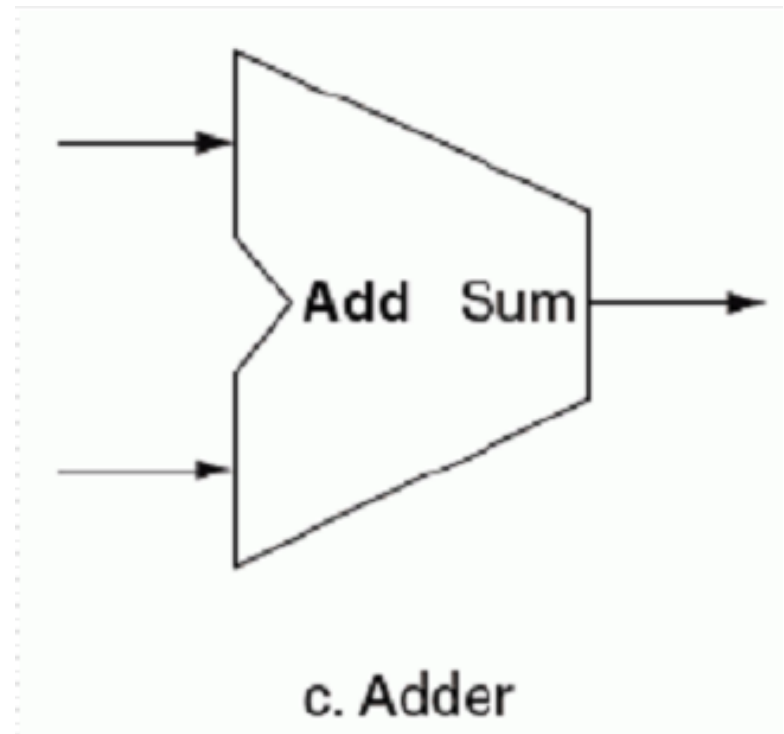
Programm counter (PC)

- 32 bitu reģistrs
- Satur aktuālās instrukcijas adresi
- Tiek pārrakstīts pēc katras operācijas

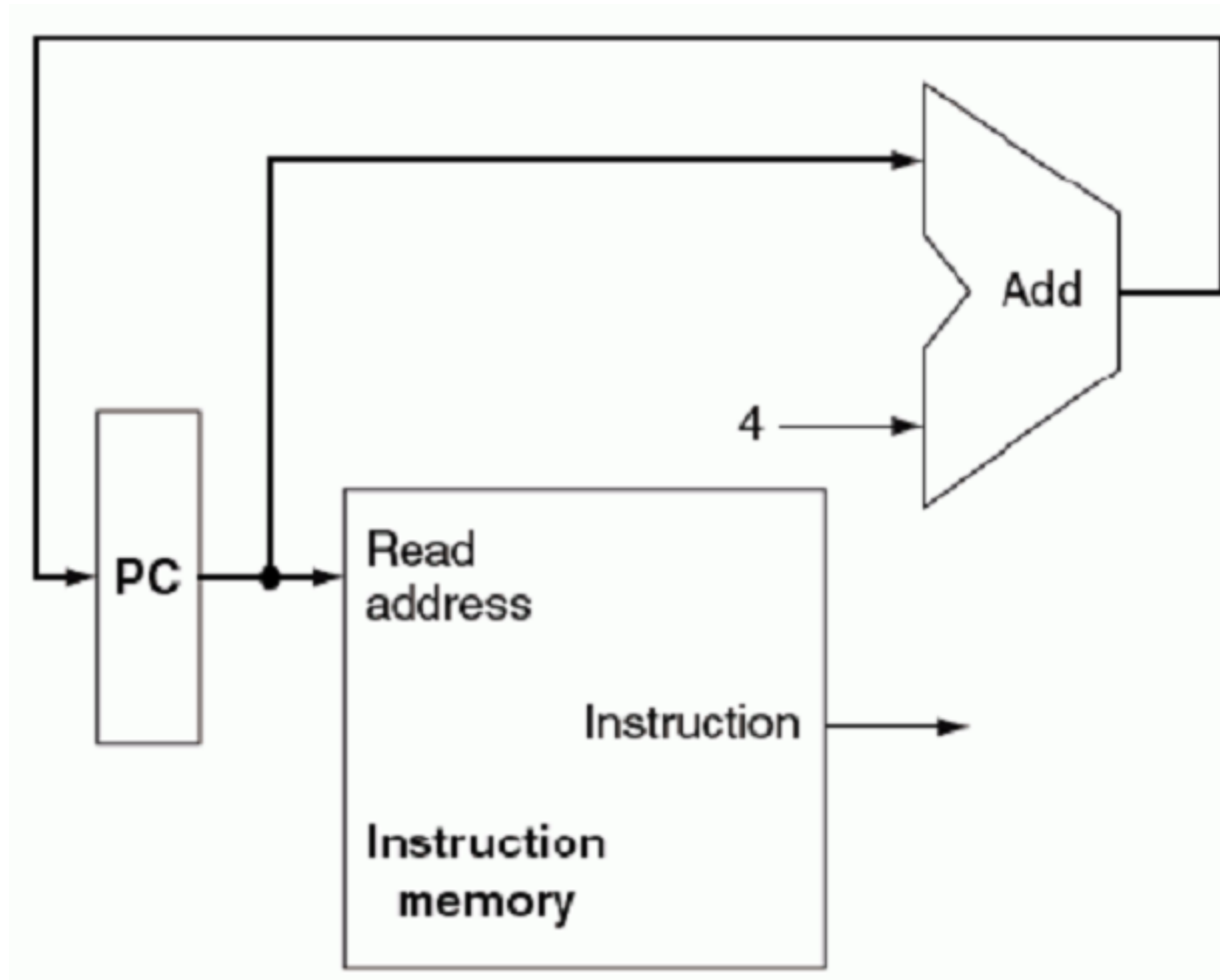


Skaitītājs (Adder)

- Nepieciešams, lai izrēķinātu nākošās instrukcijas adresi
- Skaitīšanas konstante ir atkarīga no instrukcijas garuma



Apskatītie moduļi kopskatā



**Paldies par uzmanību!
Jautājumi?**