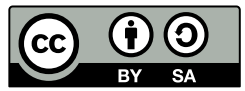
Computer architecture. An introduction

Saulius Gražulis

Vilnius, 2020

Vilnius University, Faculty of Mathematics and Informatics
Institute of Informatics



Course organisation

https://emokymai.vu.lt/course/view.php?id=15112

Assessment strategy	Weight %	Deadline	Assessment criteria
Lecture quizzes	10	10 min. at the beginning of each practical.	4-question quiz covering several recent lectures (Bloom's 1 an 2 level questions) using an electronic teaching environment (Moodle, Open edX or similar).
Intermediate quiz	15	mid-term	Approx. 30-question quiz covering several recent lectures (Bloom's 1 to 9 level questions) using an electronic teaching environment (Moodle, Open edX or similar).
Evaluation of practical assignments	50	After each practical according to the schedule announced by the teacher of the practical	The practical work schedule and evaluation criteria are announced by the teacher of each group. The teacher who leads practical work grades it and communicates the assigned grade.
Analysis of an assigned computer architecture example	10	end of term	Students provide a written (4 pages, A4 format, 9pt) technical report on a computer architecture which they studied themselves, or 5 min oral presentation with slides on that architecture. The oral presentation is available for achieving students, with permission of the teaching professor, and may be accepted as a final exam.
Final exam	15	end of term	approx. 30-question quiz covering several recent lectures (Bloom's 1 to 9 level questions) using an electronic teaching environment (Moodle, Open edX or similar).

Course exam requirements

To be eligible for the exam, students must fulfil all following criteria:

1. carry out at least one practical work and get a positive grade for the practicals;
2. have enough accumulated points to be able to pass the exam in principle if they score maximum points at the exam quiz;
3. students who have shown excellent performance during the term may be freed from the final exam quiz and given the opportunity to present their research on computer architectures as the oral presentation, provided:
    1. collect at least 60% of all possible points during the theory course (e.g. at least 150 points from the 250 possible points from the intermediate exam and the lecture quizzes);
    2. they have carried out all practical assignments in the scheduled time;
    3. they have positive recommendations from the teacher of their practical team.

If there are more students wishing to make oral presentations than the allocated time permits, students with higher grades will be given priority.

Course exam requirements

Course description

			Unless you are allowed to take the oral exam (i.e. make the presentation), participation in the final exam quiz is obligatory to pass the course, regardless of the accumulated points. Students who do not show up in the final exam will be indicated as such in the exam grading report. To pass the exam, on must score at least 50% of possible points.
Total	100		The final mark is obtained summing up all points obtained for each task, quiz or assignment, dividing them by 100 and rounding to the *higher* integer (i.e. 0.001 is rounded to 1.0; 9.1 is rounded to 10).

Course structure

- First half: understanding of internal functions of a CPU and a computer on the level of logic circuits; modelling in Logisim;
- Second half: acquaintance with some widespread and self-made computer architectures and assembler programming;

Rules of video conferencing

- Only the speaker has the microphone ON; all others must switch their microphones **OFF**;
- The speaker **should** have camera on (VU rules!);
- When a teacher asks you a question, you **must** raise your hand and, when assigned a talk slot, switch on you microphone and answer; or type your answer in the chat, as requested (VU rules!);
- To ask a question, use the “raise hand” functionality and/or post your question to the forum;
- Recordings of lectures are copyrighted; it is **not permitted** to post them outside the university without consent of the lecturer and university officials;
- If the connection is interrupted, **do not leave** the lecture – the teacher will set up alternative connection in several minutes;

Why study computer architecture?

> *Most computer users have an incorrect, but useful, cognitive metaphor for computers in which the user says (or types or clicks) something and a mystical, almost intelligent or magical, behavior happens.*
>
> Charles W. Kann (2016)

- We must understand architecture if we want to program it efficiently
- Improves our education – we finally understand how **these** things work!
- It is a terribly interesting subject :)

What is a computer?

- Computer is **digital**:

Emural screenshot

- Computer is **automatic**:

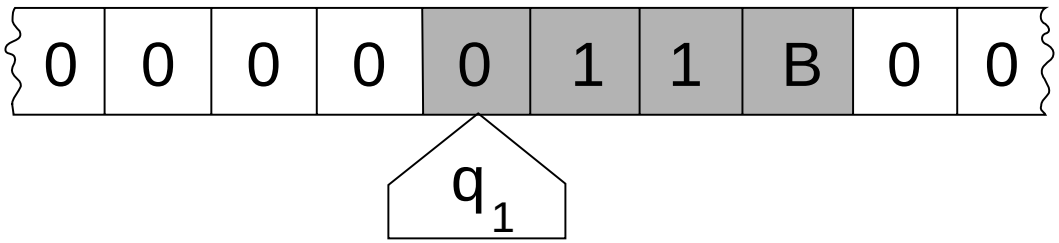
Wikipedia: Skylab. Image by NASA, public domain.

What is a computer?

- Computer can be **reprogrammed**:

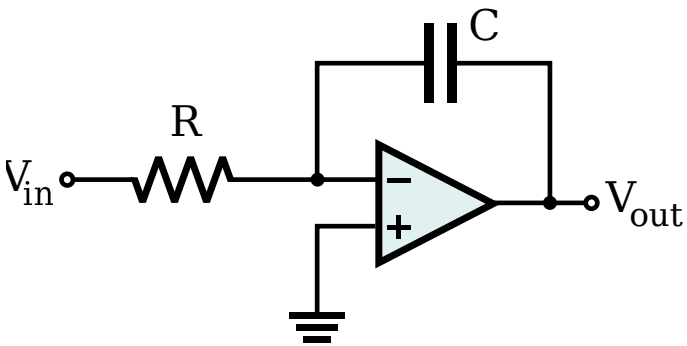
```
#! /usr/bin/perl

use strict;
use warnings;

my $selected_line;

while( <> ) {
    $selected_line = $_ if rand() < 1/$.;
}

print $selected_line;
```

- Computer can solve **any** task that is feasible **mathematically** (Turing 1937; Wikipedia 2020):



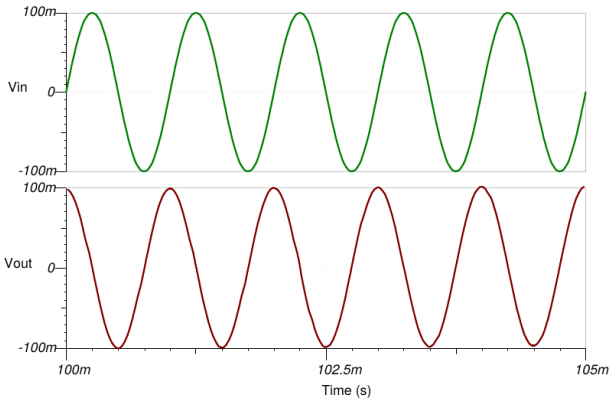
Nynexman4464 [*CC-BY-SA 3.0*]

Side-note: Analog computers



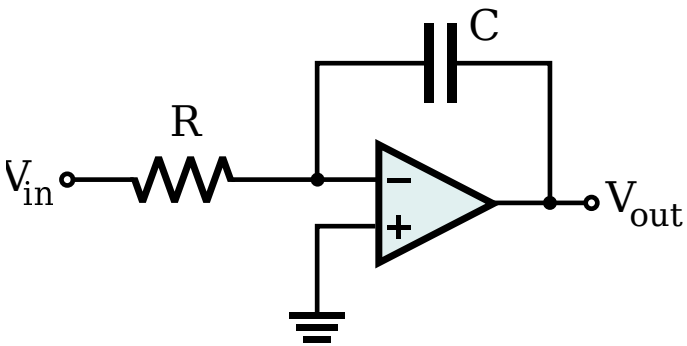
Inductiveload [Public domain]

$$V_{\text{out}}(t) = -\frac{1}{RC}\int_0^t V_{\text{in}}(t)dt$$



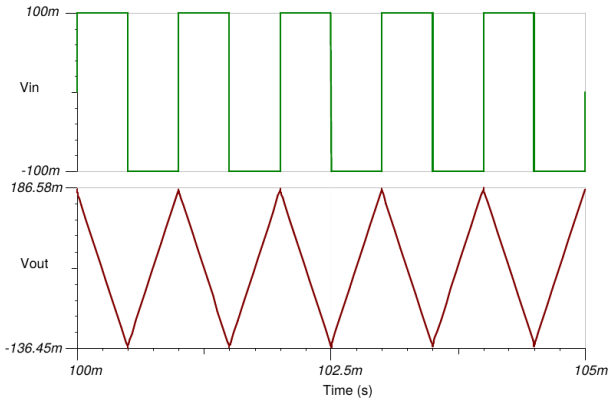
(Texas Instruments 2019)

Side-note: Analog computers



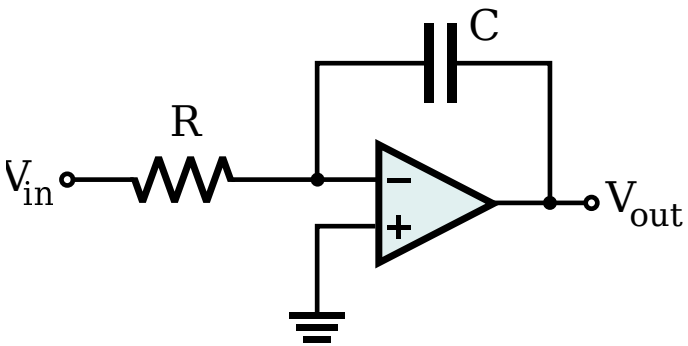
Inductiveload [Public domain]

$$V_{\text{out}}(t) = -\frac{1}{RC}\int_0^t V_{\text{in}}(t)dt$$



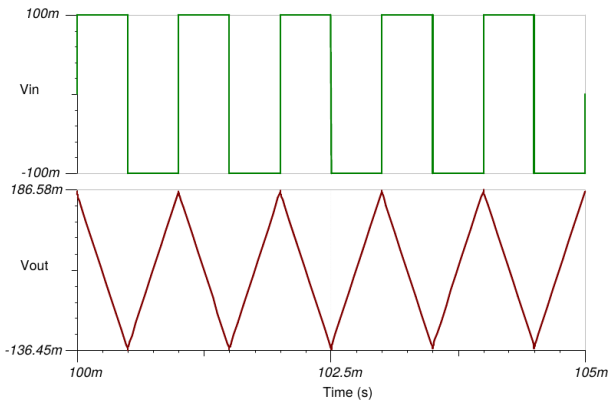
(Texas Instruments 2019)

Side-note: Analog computers

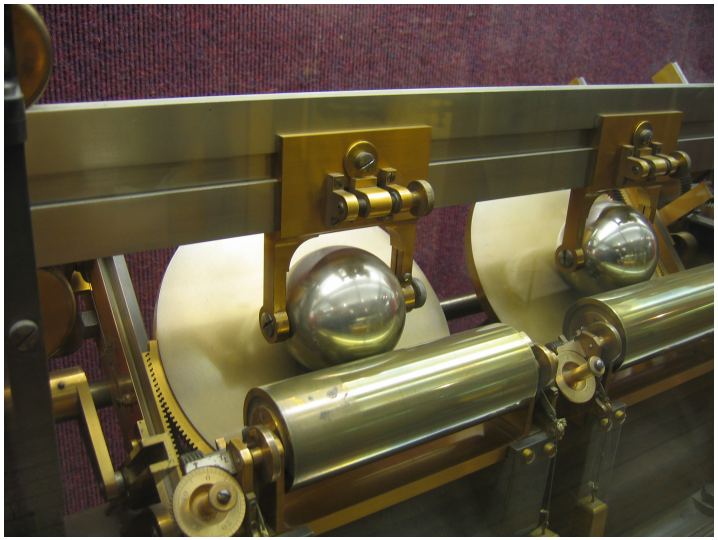


Inductiveload [Public domain]

$$V_{\text{out}}(t) = -\frac{1}{RC}\int_0^t V_{\text{in}}(t)dt$$

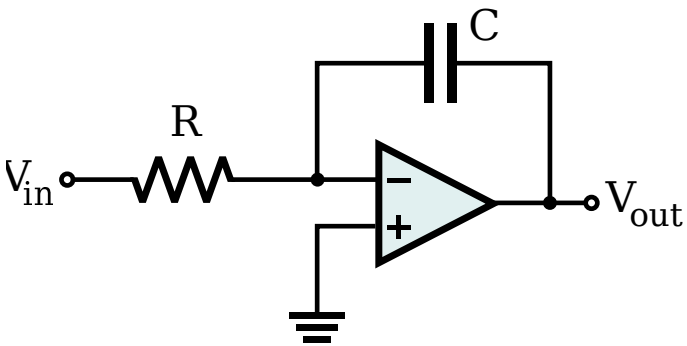


(Texas Instruments 2019)



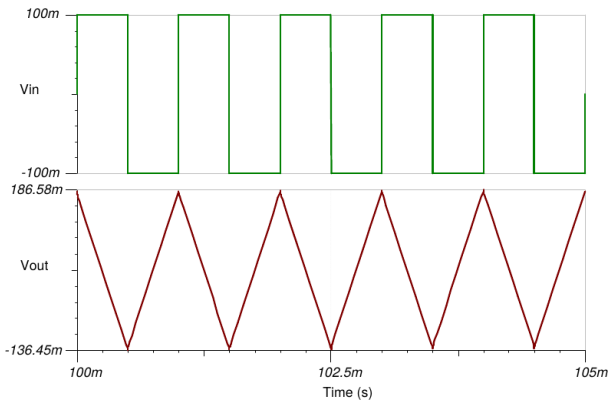
Tide-predicting machine, around 1878
Andy Dingley [CC BY 3.0]

Side-note: Analog computers

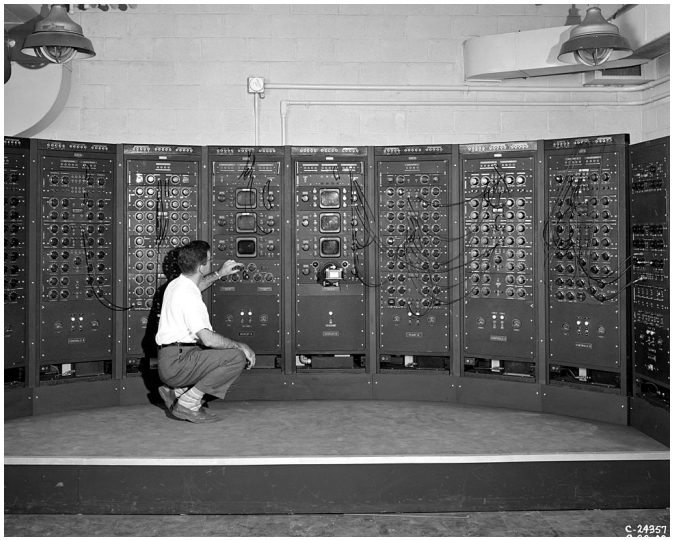


Inductiveload [Public domain]

$$V_{\text{out}}(t) = -\frac{1}{RC}\int_0^t V_{\text{in}}(t)dt$$

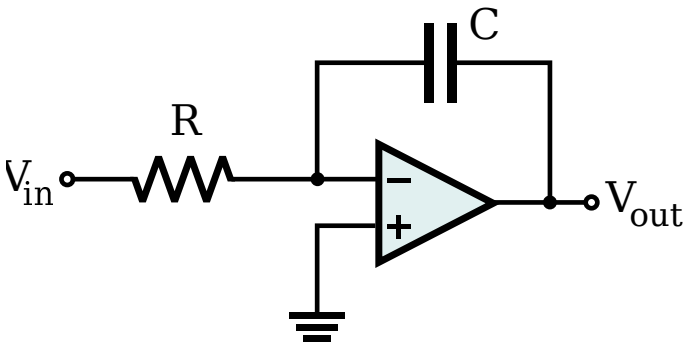


(Texas Instruments 2019)



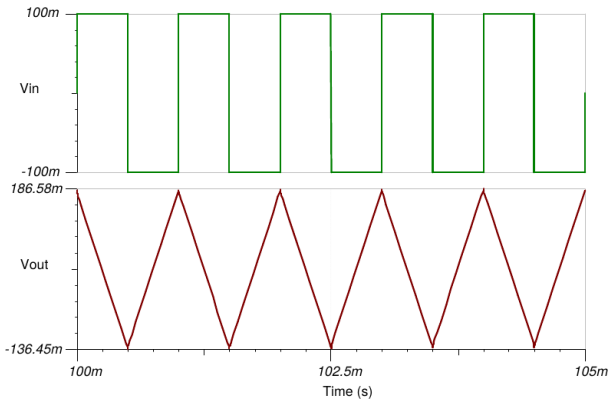
Electronic Analog computer, circa 1949
NASA [Public domain]

Side-note: Analog computers

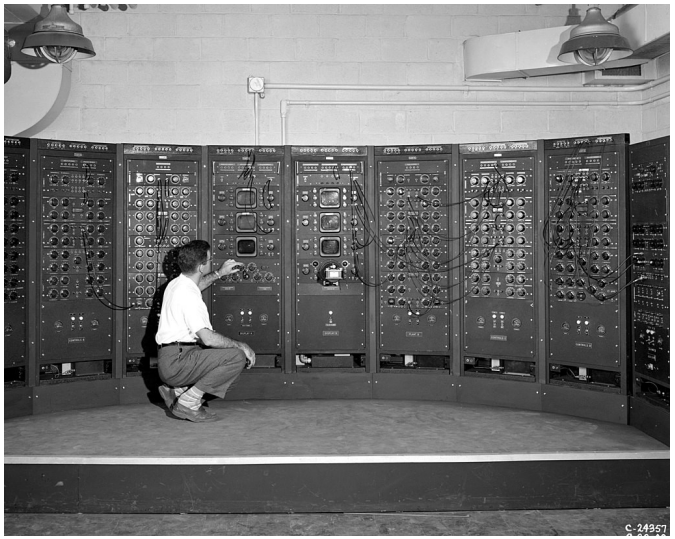


Inductiveload [Public domain]

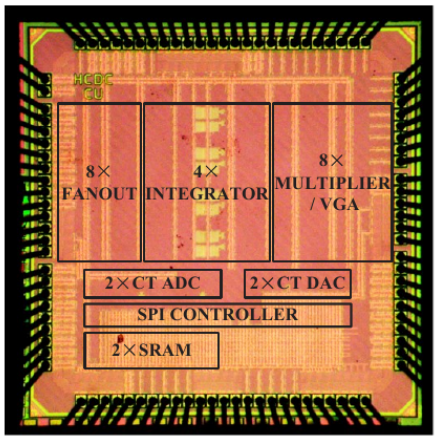
$$V_{\text{out}}(t) = -\frac{1}{RC}\int_0^t V_{\text{in}}(t)dt$$



(Texas Instruments 2019)

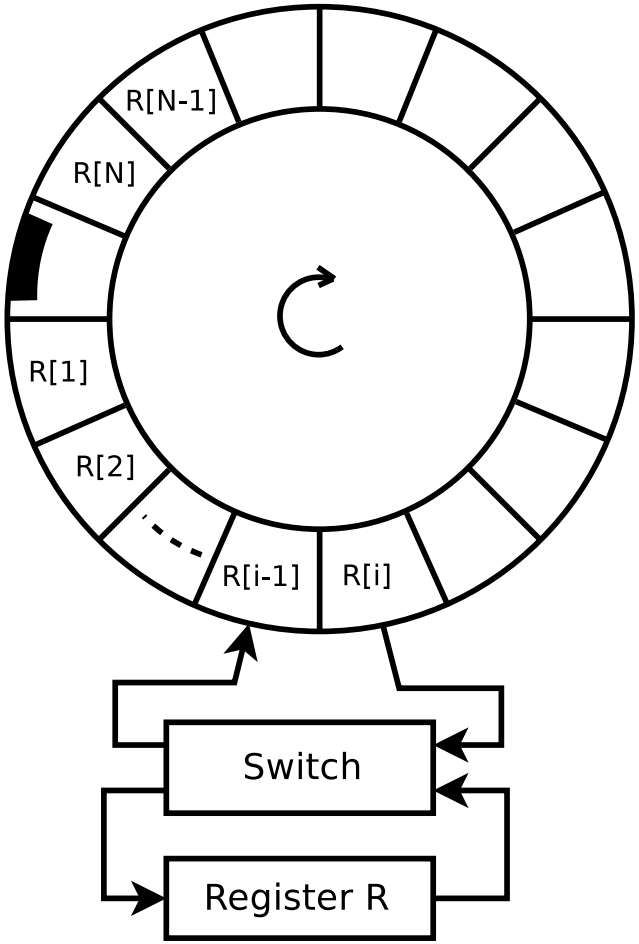


Electronic Analog computer, circa 1949
NASA [Public domain]



(Guo et al. 2015)

“Bubble-sort” machine



Step 1. Set $R \leftarrow R_1$.
(R is an internal machine register.)

Step i. for $1 < i \leq N$:

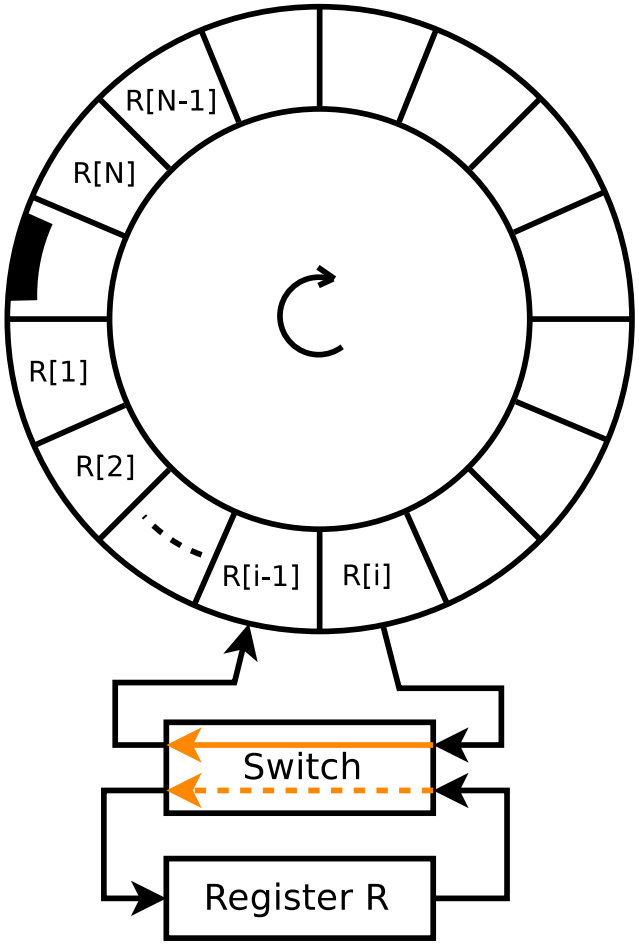
Either

i. set $R_{i-1} \leftarrow R$, $R \leftarrow R_i$, or
ii. set $R_{i-1} \leftarrow R_i$, leaving R unchanged.

Step $N+1$. Set $R_N \leftarrow R$.

(Knuth 1997), ch. 5.3.4 (p. 246)

“Bubble-sort” machine



Step 1. Set $R \leftarrow R_1$.
(R is an internal machine register.)

Step i. for $1 < i \leq N$:

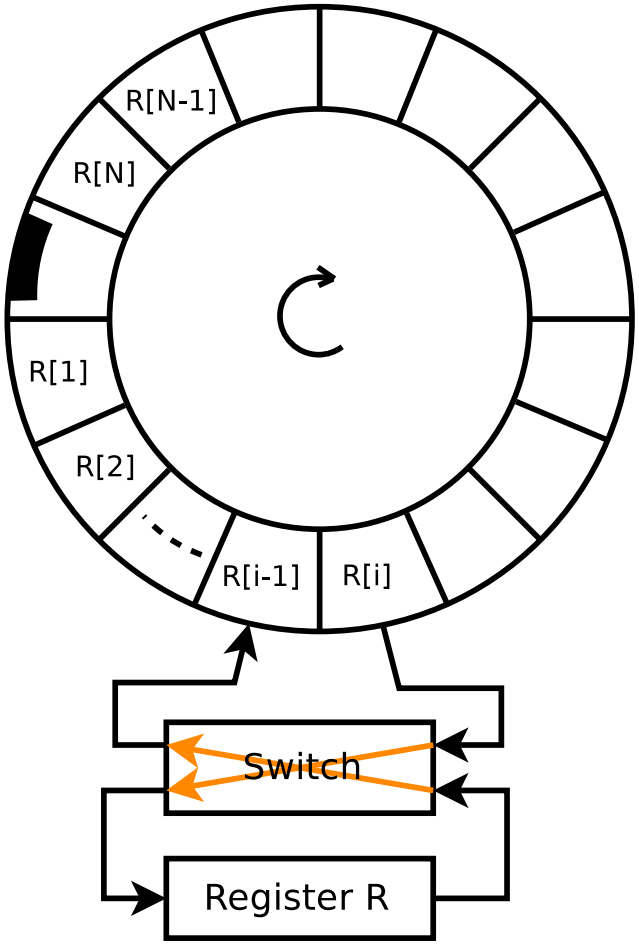
Either

i. set $R_{i-1} \leftarrow R$, $R \leftarrow R_i$, or
ii. set $R_{i-1} \leftarrow R_i$, leaving R unchanged.

Step $N+1$. Set $R_N \leftarrow R$.

(Knuth 1997), ch. 5.3.4 (p. 246)

“Bubble-sort” machine



Step 1. Set $R \leftarrow R_1$.
(R is an internal machine register.)

Step i. for $1 < i \leq N$:

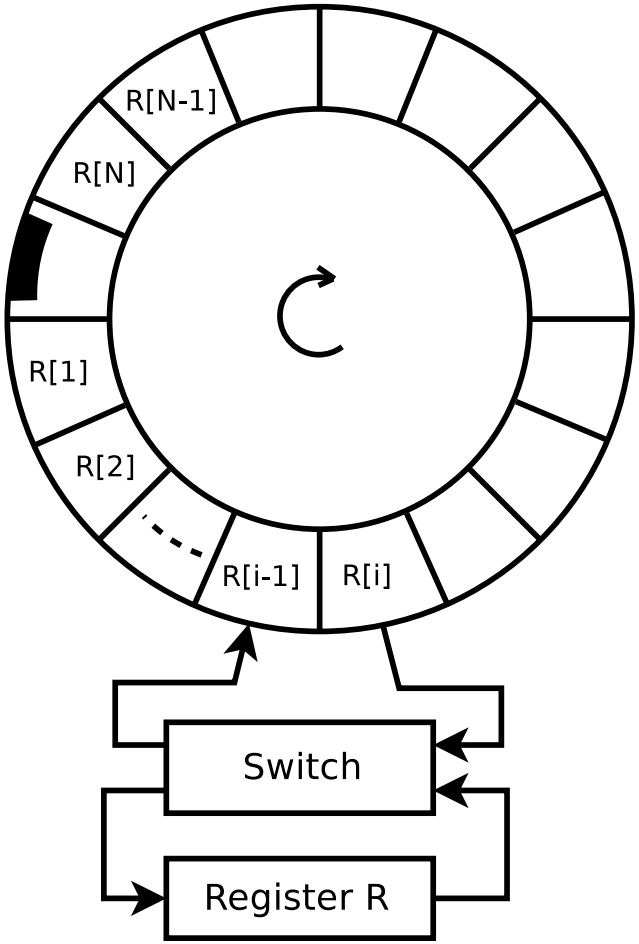
Either

i. set $R_{i-1} \leftarrow R$, $R \leftarrow R_i$, or
ii. set $R_{i-1} \leftarrow R_i$, leaving R unchanged.

Step $N+1$. Set $R_N \leftarrow R$.

(Knuth 1997), ch. 5.3.4 (p. 246)

“Bubble-sort” machine



(Knuth 1997), ch. 5.3.4 (p. 246)

Step 1. Set $R \leftarrow R_1$.
(R is an internal machine register.)

Step i. for $1 < i \leq N$:

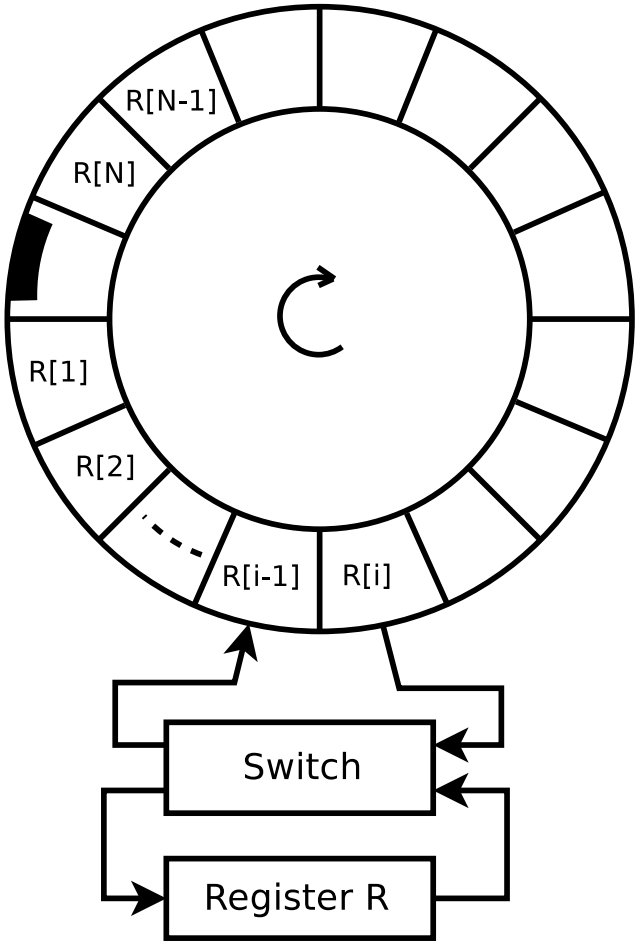
Either

i. set $R_{i-1} \leftarrow R, R \leftarrow R_i$, or
ii. set $R_{i-1} \leftarrow R_i$, leaving R unchanged.

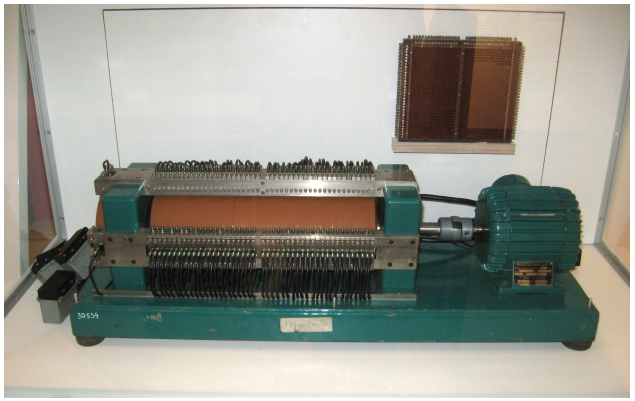
Step $N+1$. Set $R_N \leftarrow R$.

It turns out that for this automaton, the bubble sort algorithm is optimal! (Howard B. Demuth, PhD Thesis, 1956)

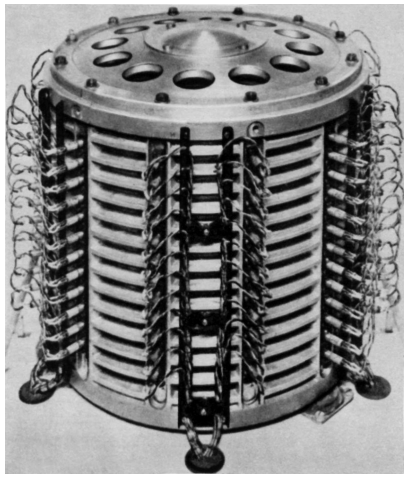
“Bubble-sort” machine



(Knuth 1997), ch. 5.3.4 (p. 246)



Wikipedia: the BESK drum memory



Wikipedia: the Polish ZAM-41 drum memory

Float number comparisons

```c
#include <stdio.h>
#include <math.h>

int main()
{
    float a = 0.0, b = 0.0;
    float ratio = a/b;

    printf( "%f\n", ratio );
    printf( "%f\n", a );

    if( a <= ratio ) {
        printf( "YES\n" );
    } else {
        printf( "NO\n" );
    }

    if( a > ratio ) {
        printf( "YES\n" );
    } else {
        printf( "NO\n" );
    }

    return 0;
}
```

Float number comparisons

```c
#include <stdio.h>
#include <math.h>

int main()
{
    float a = 0.0, b = 0.0;
    float ratio = a/b;

    printf( "%f\n", ratio );
    printf( "%f\n", a );

    if( a <= ratio ) {
        printf( "YES\n" );
    } else {
        printf( "NO\n" );
    }

    if( a > ratio ) {
        printf( "YES\n" );
    } else {
        printf( "NO\n" );
    }

    return 0;
}
```

```
+ ./float-comparisons
-nan
0.000000
NO
NO
```

Float number comparisons

```c
#include <stdio.h>
#include <math.h>

int main()
{
    float a = 0.0, b = 0.0;
    float ratio = a/b;

    printf( "%f\n", ratio );
    printf( "%f\n", a );

    if( a <= ratio ) {
        printf( "YES\n" );
    } else {
        printf( "NO\n" );
    }

    if( a > ratio ) {
        printf( "YES\n" );
    } else {
        printf( "NO\n" );
    }

    return 0;
}
```

```
+ ./float-comparisons
-nan
0.000000
NO
NO
```

Thus, for floating point number comparisons the following statement does **not** hold:

$$\neg(a \leq b) \Rightarrow a > b$$

“Patriot” missile failure...

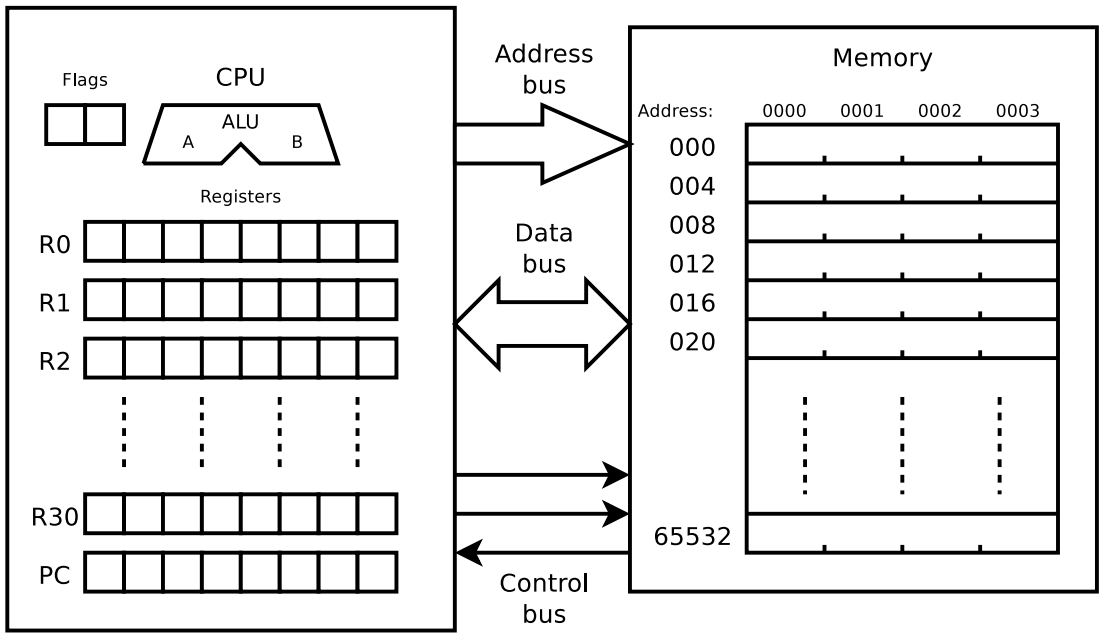
American Patriot Missile

[CC-BY-SA]

- The machine (the computer) was **binary**;
- The time was stored in a **24 bit** register;
- The time was measured by the system in **1/10 second** units;
- The number 1/10 in decimal is **infinite periodic** binary fraction – not representable exactly in binary;
- The rounding error was:
  $0.1_{10} - 0.0001\,1001\,1001\,1001\,1001\,1000_2 = 9.5 \times 10^{-08}_{10}$
- In 100 h, this accumulates:
  $9.5 \times 10^{-08} ds \cdot 10 \frac{ds}{s} \cdot 3600 \frac{s}{h} \cdot 100h = \mathbf{0.34s}$
- During **0.34s** the incoming Scud missile travels more than half a kilometre and is missed by the Patriot system...

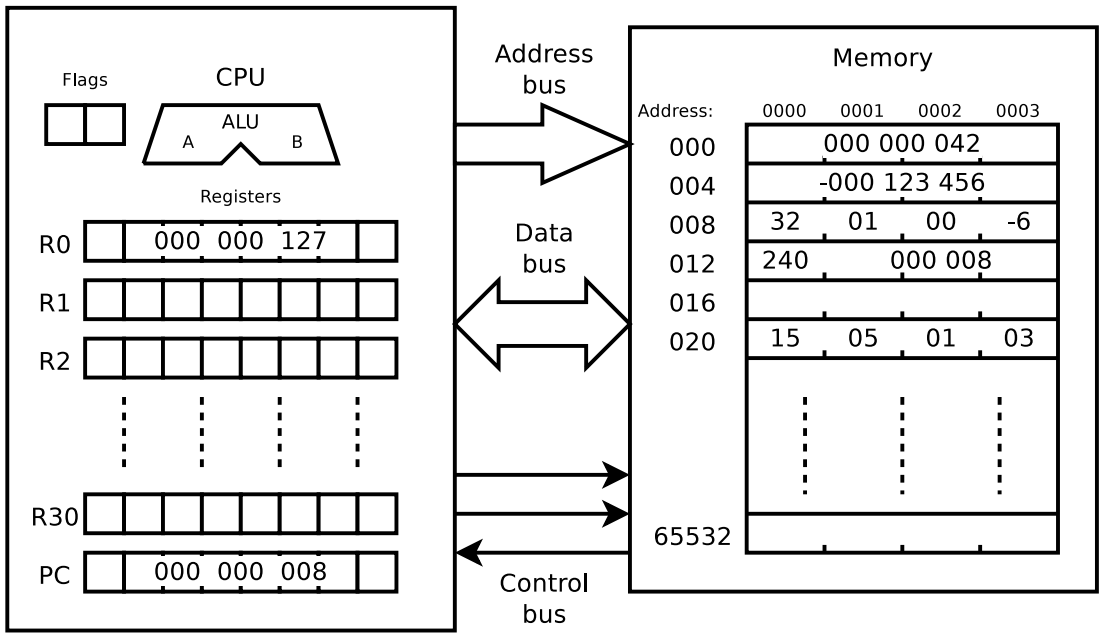
An overview of a computer

Von Neumann architecture

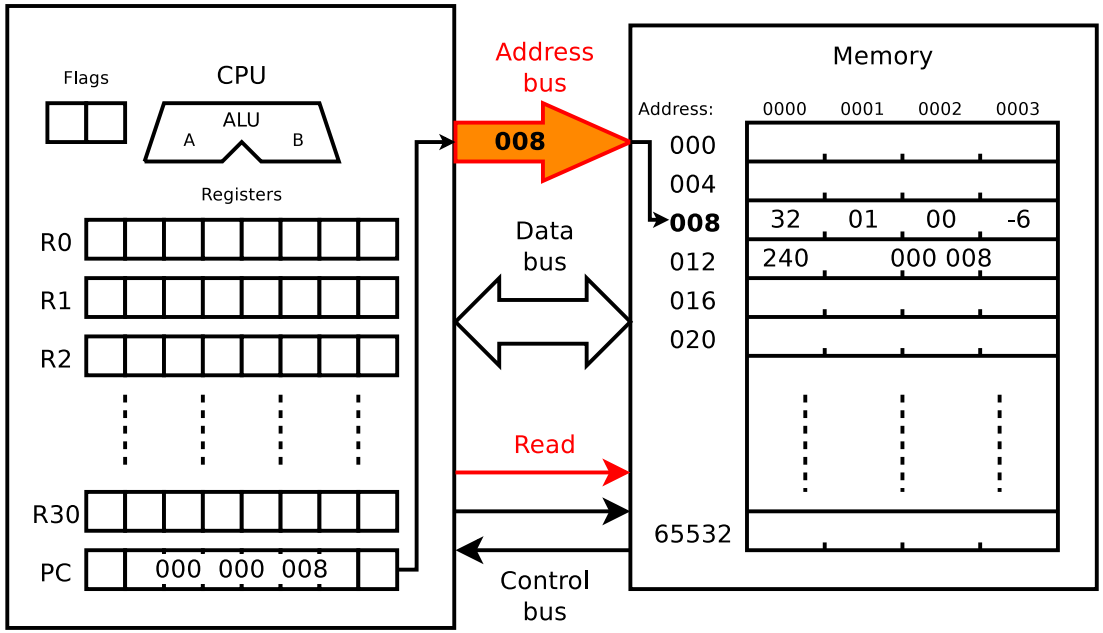


An overview of a computer

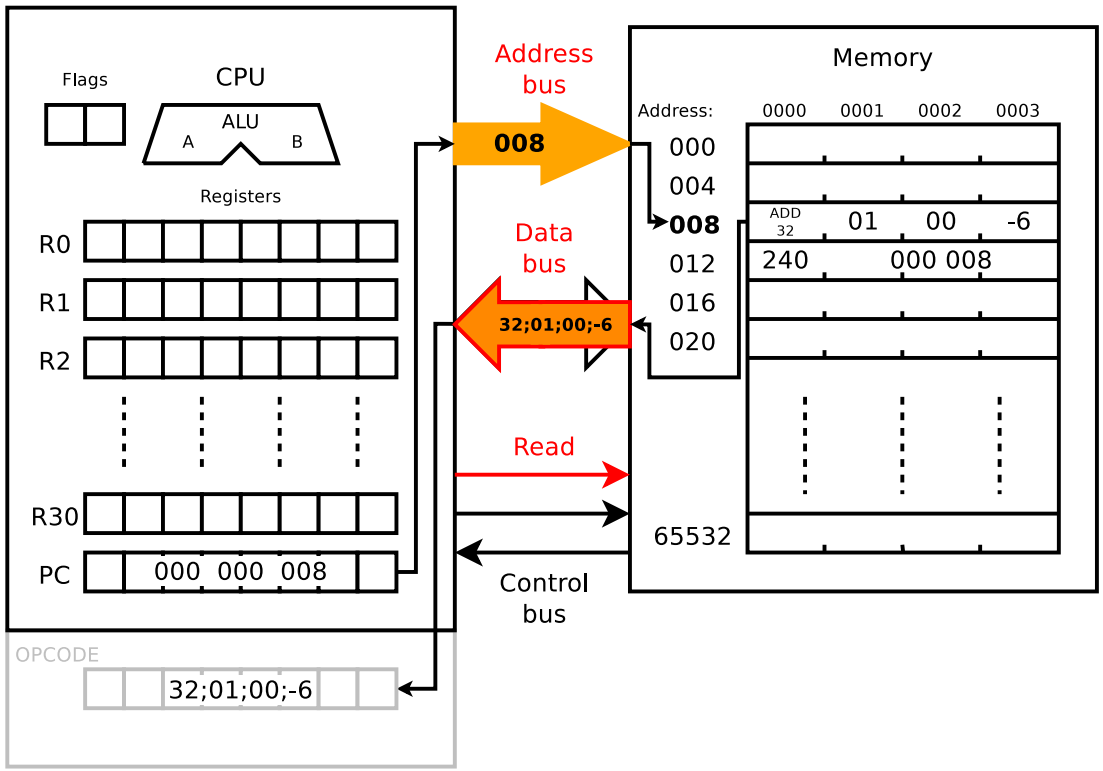
Von Neumann architecture



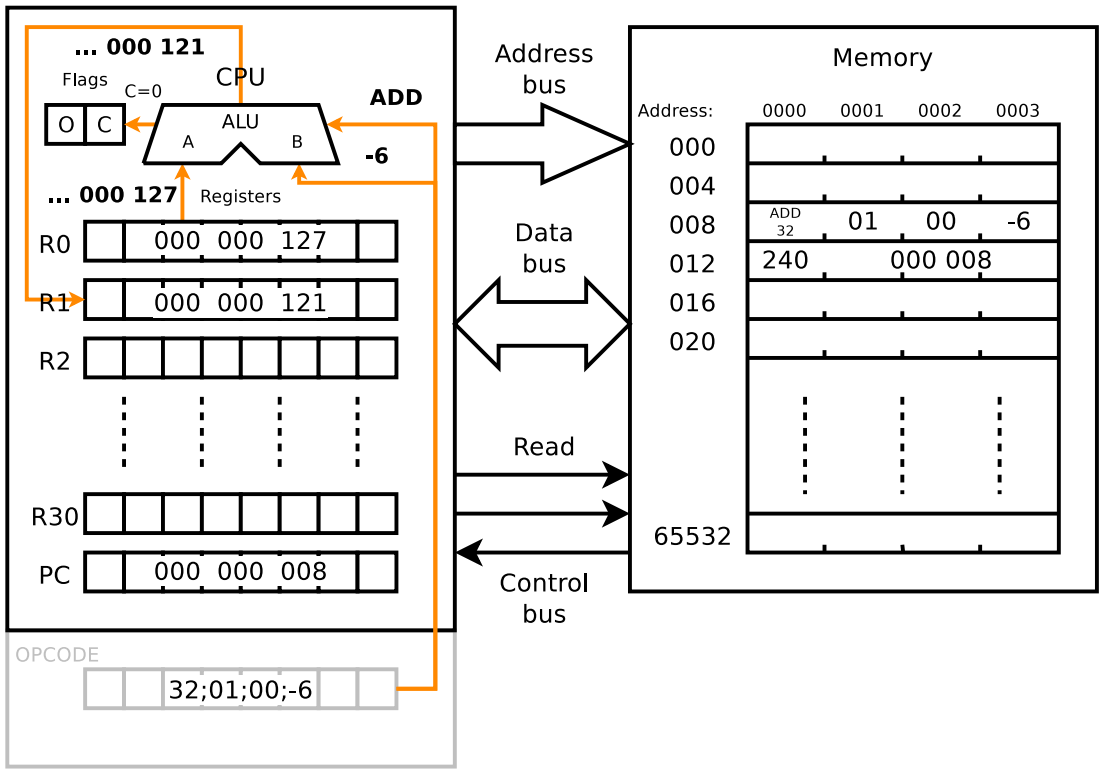
Command execution cycle



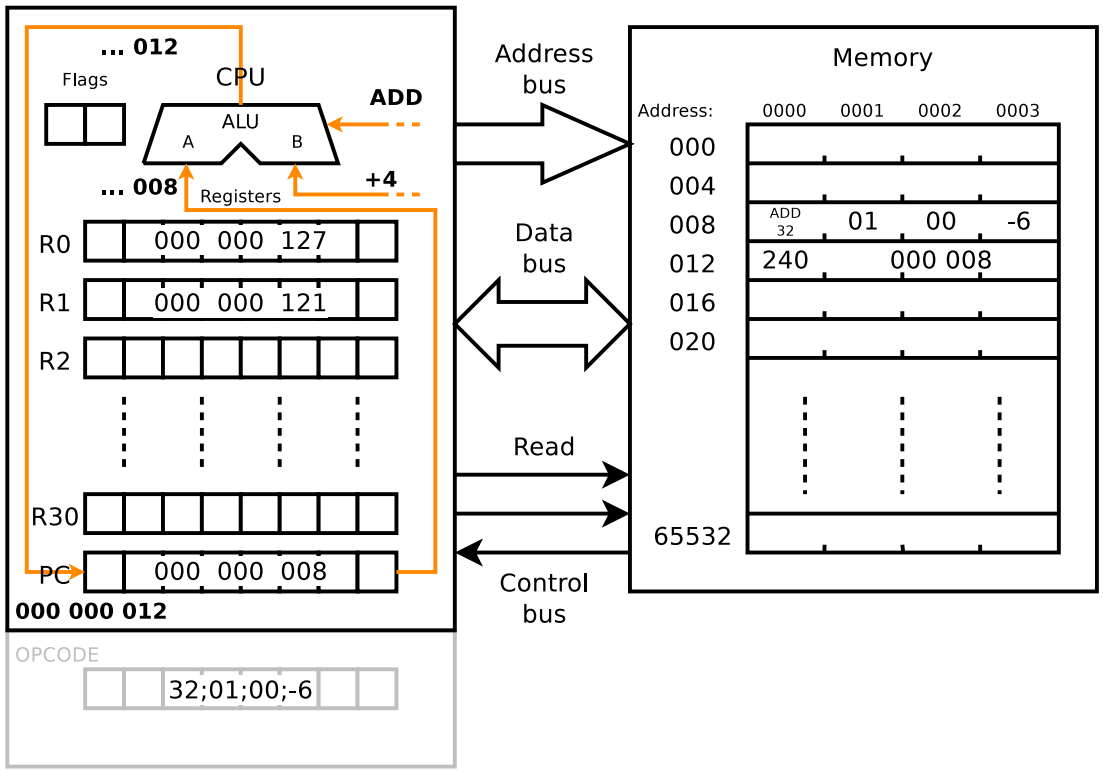
Command execution cycle



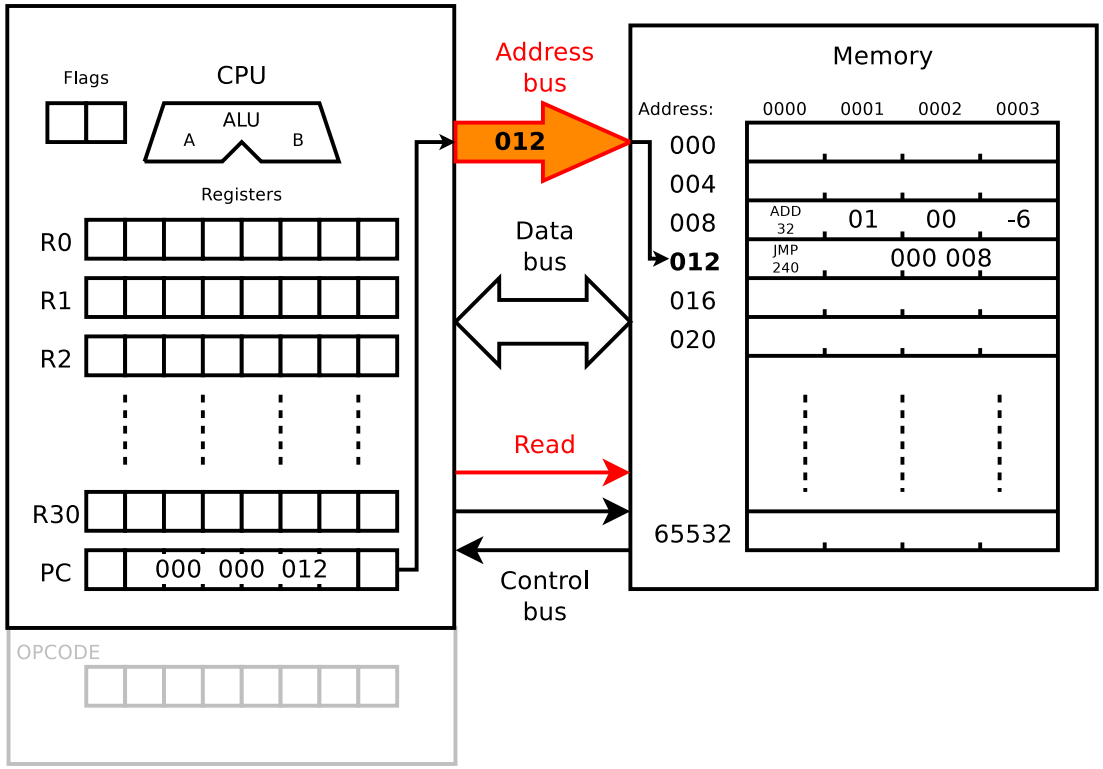
Command execution cycle



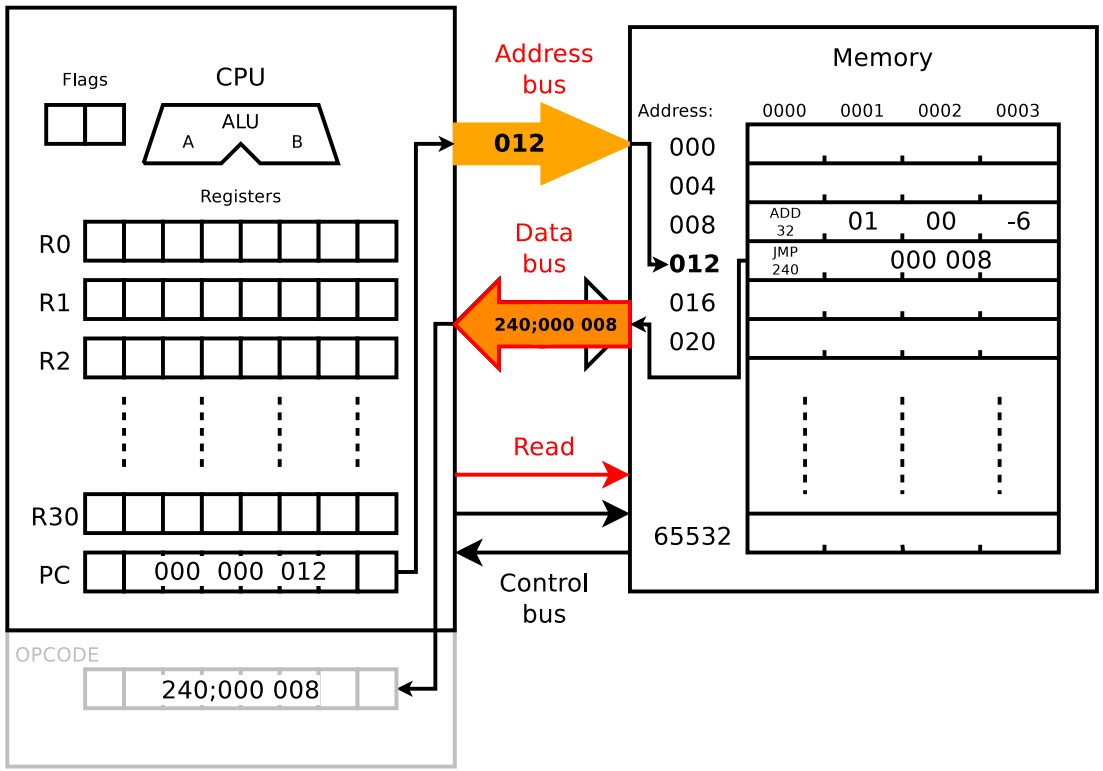
Command execution cycle



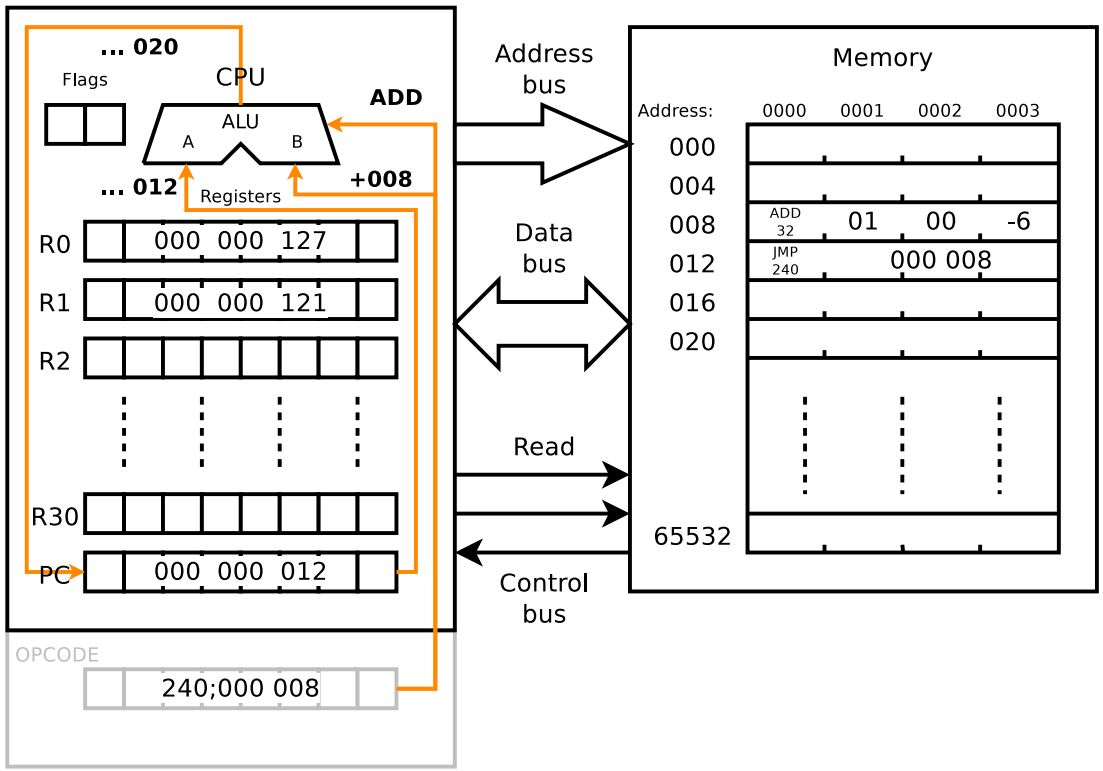
JMP instruction



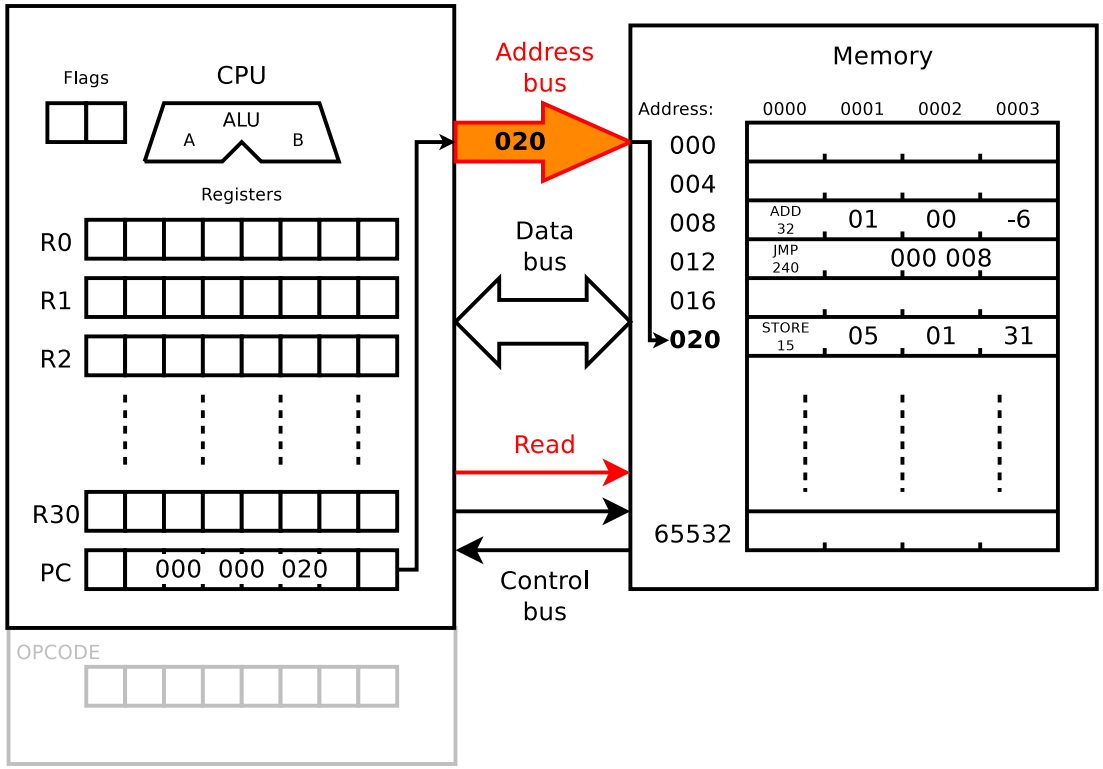
JMP instruction



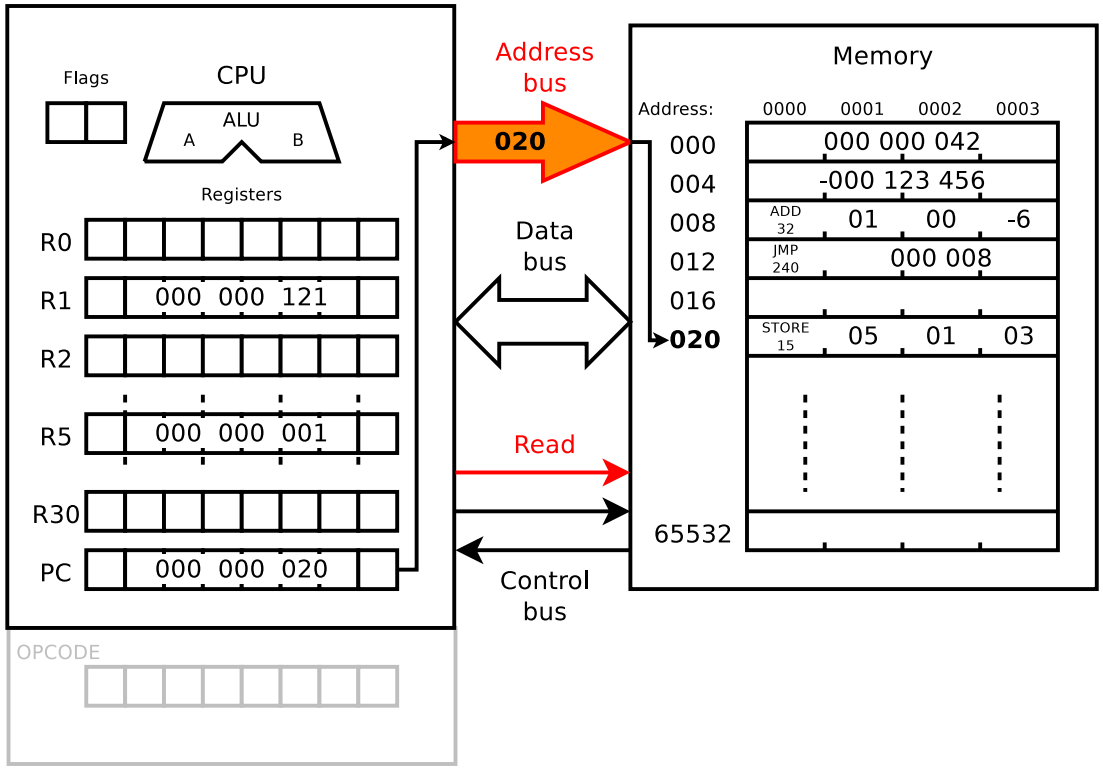
JMP instruction



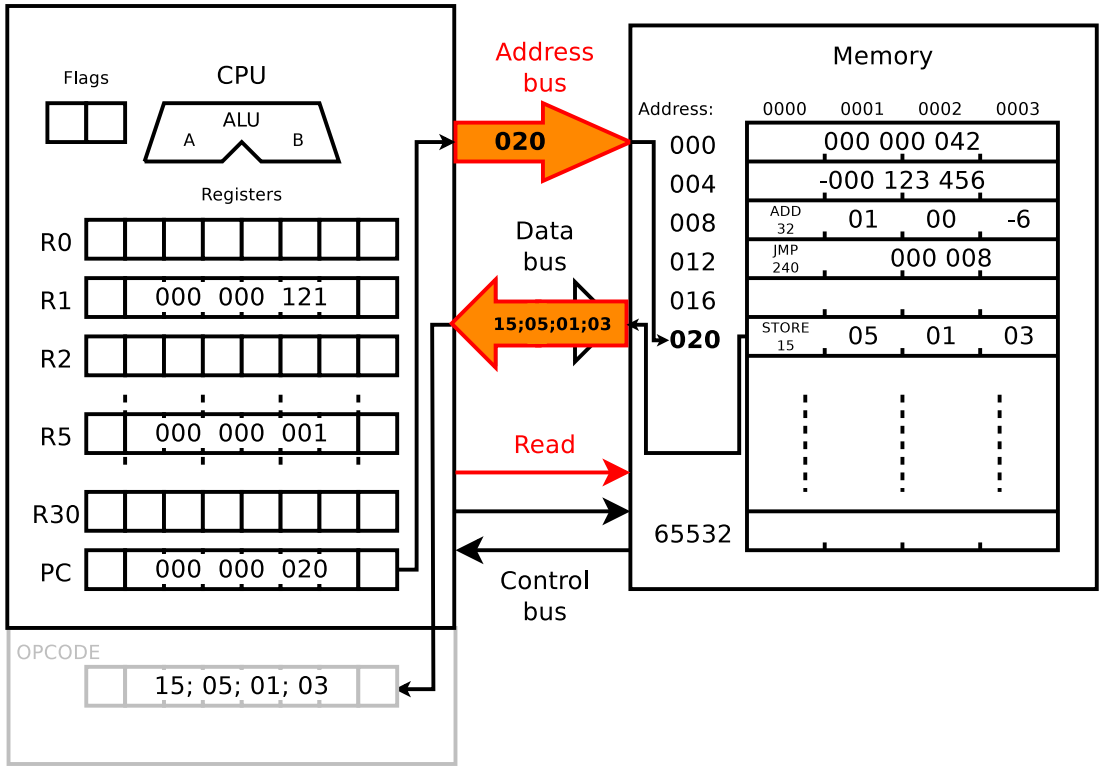
JMP instruction



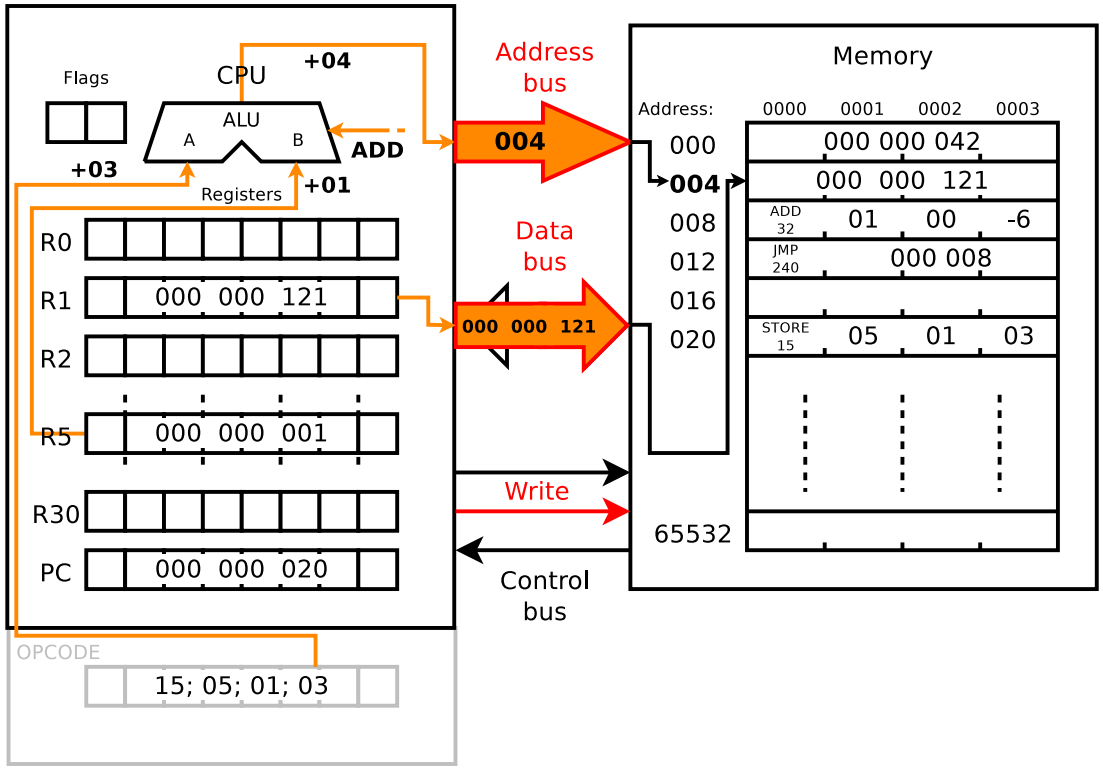
STORE instruction



STORE instruction

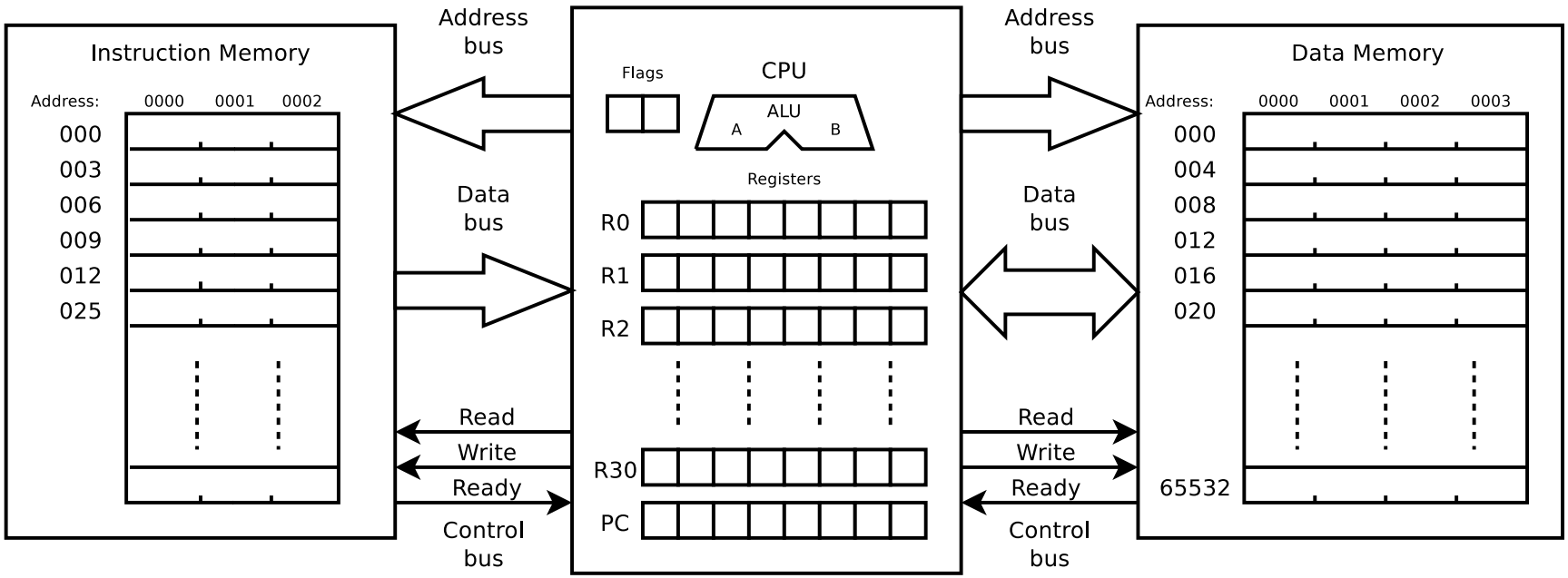


STORE instruction



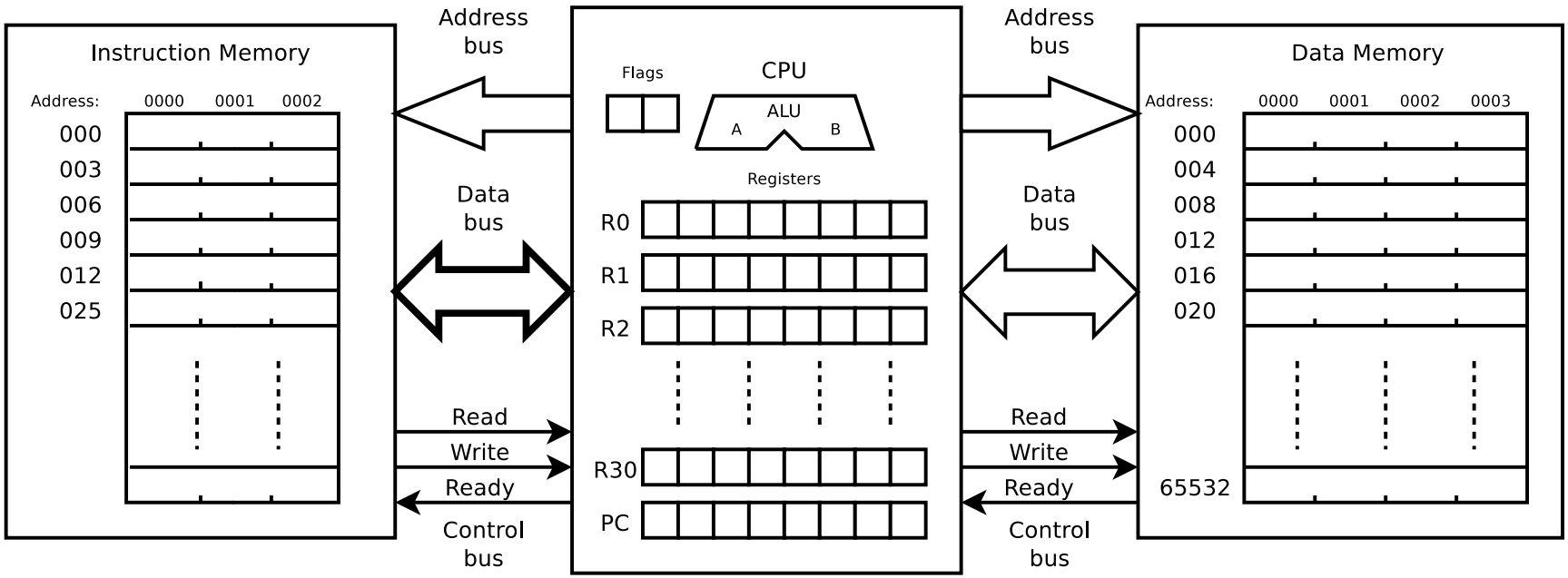
Harvard architecture

Classic Harvard architecture

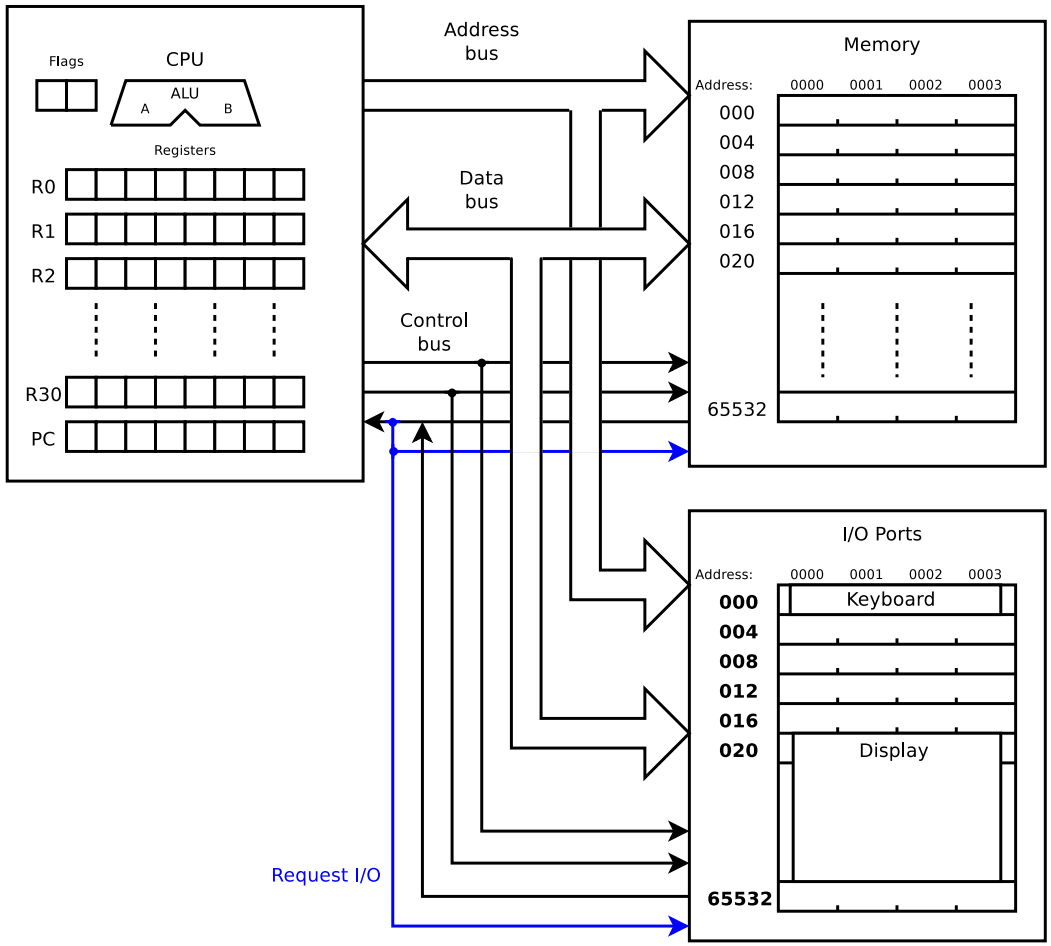


Harvard architecture

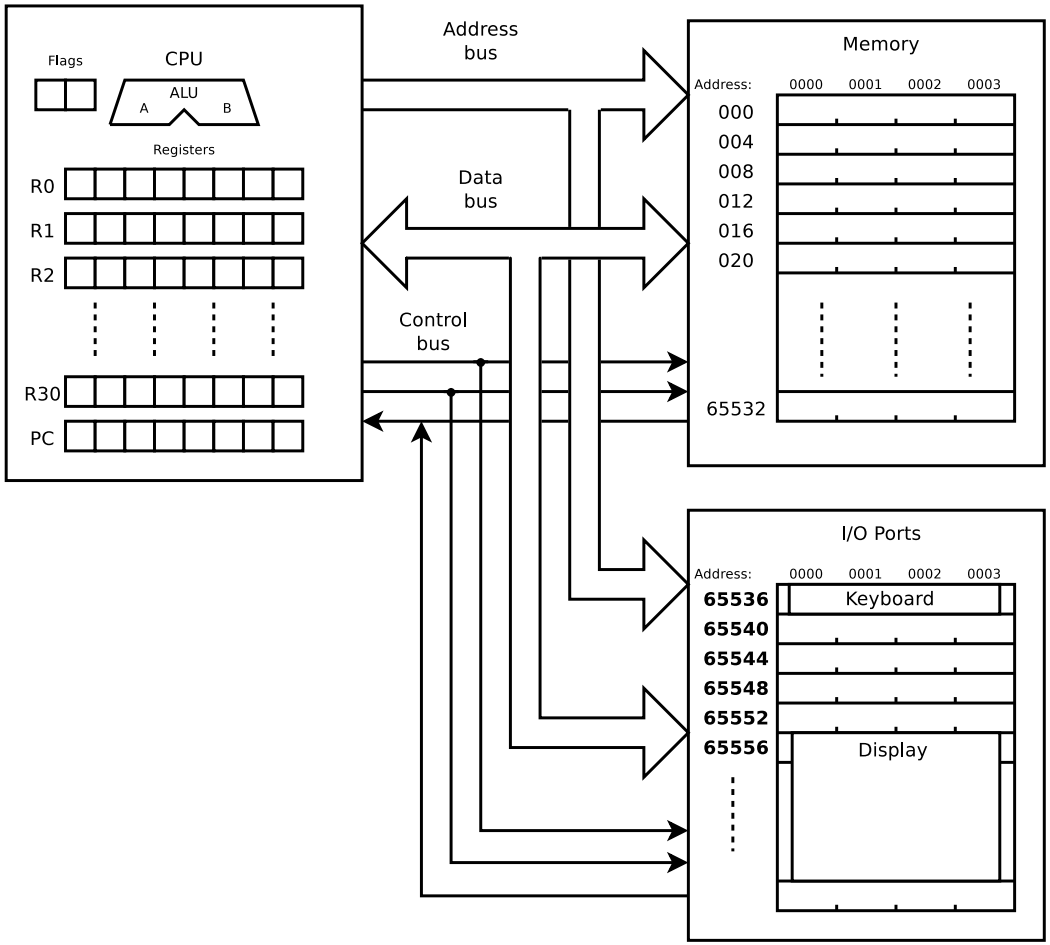
Modified Harvard architecture



Separate I/O address space



Memory-mapped I/O

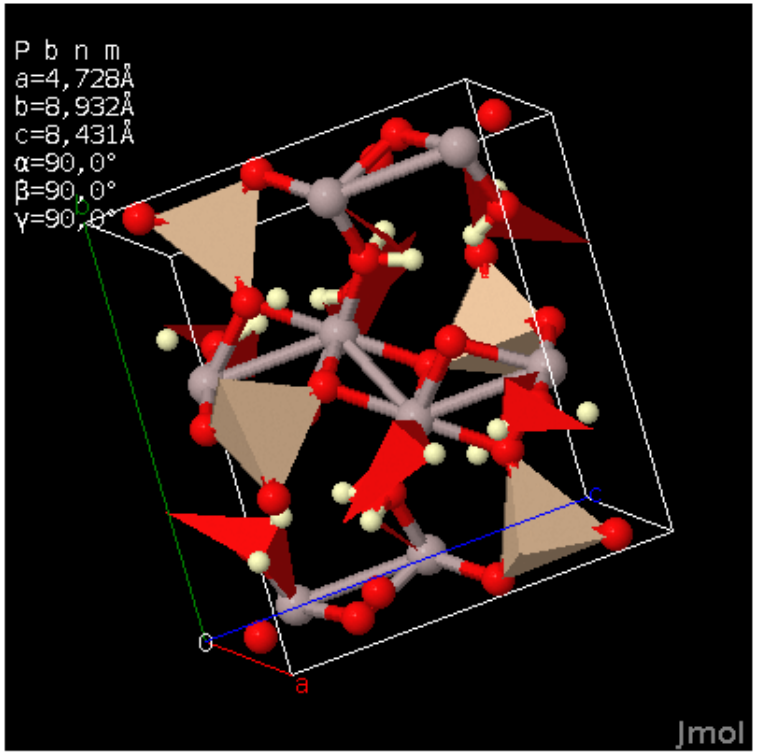


Take home messages

- Digital computers are automatic, programmable, universal devices (but analog computers also exist(ed));
- Knowing computer architecture is important if we want to program them correctly and efficiently;
- Various architectures exists (von Neumann, Harvard), but in all cases two main components – CPU and memory – are essential;
- The CPU fetches instructions from memory and performs various operations on data in memory and in registers, as indicated by instructions;

Questions?

http://en.wikipedia.org/wiki/Topaz



Coordinates	2207377.cif
Original IUCr paper	HTML

http://www.crystallography.net/2207377.html

References

Demuth, Howard B. (Oct. 1956). "Electronic Data Sorting". PhD thesis. Stanford University.

Guo, Ning et al. (Sept. 2015). "Continuous-time hybrid computation with programmable nonlinearities". In: *ESSCIRC Conference 2015 - 41st European Solid-State Circuits Conference (ESSCIRC)*. IEEE, pp. 279–282. DOI: 10.1109/esscirc.2015.7313881.

Kann, Charles W. (2016). *Implementing a One Address CPU in Logisim*. WWW: https://open.umn.edu/opentextbooks/textbooks/implementing-a-one-address-cpu-in-logisim. Gettysburg College. URL: https://open.umn.edu/opentextbooks/formats/989.

Knuth, Donald E. (July 1997). *The art of computer programming. Sorting and searching*. Vol. 3. Addison-Wesley. ISBN: 978-02-0189-685-5.

Texas Instruments (2019). *Integrator circuit*. Tech. rep. Texas Instruments, pp. 1–6. URL: https://www.ti.com/lit/an/sboa275a/sboa275a.pdf.

Turing, A. M. (1937). "Computability and λ-definability.". English. In: *J. Symb. Log.* 2, pp. 153–163. ISSN: 0022-4812; 1943-5886/e. DOI: 10.2307/2268280.

Wikipedia (2020). *Turing machine*. URL: https://en.wikipedia.org/wiki/Turing_machine.