NTNU – Trondheim
Norwegian University of
Science and Technology

The von Neumann computer

Jan.Christian.Meyer@ntnu.no

Some background

- Our goal today is to establish some terminology to discuss parallel computer design
- You may recognize the elements from other classes on computer fundamentals, but I’ll repeat them to make sure that we’re all on the same page
- I’ll also simplify them quite a bit, in order to establish an appropriate level of abstraction that will allow us to finish this course before Christmas

Start with (main) memory

- This is its own separate circuitry, its only job is to receive table indices and react accordingly

What do you have at 00000101?

That is 00100000

Address	Contents
00000000	01001000
00000001	01100101
00000010	01101100
00000011	01101100
00000100	01101111
00000101	00100000
00000110	00000000

Start with (main) memory

- We can also overwrite entries in the table:

Put this in

at 00000110!

The number is 01110111

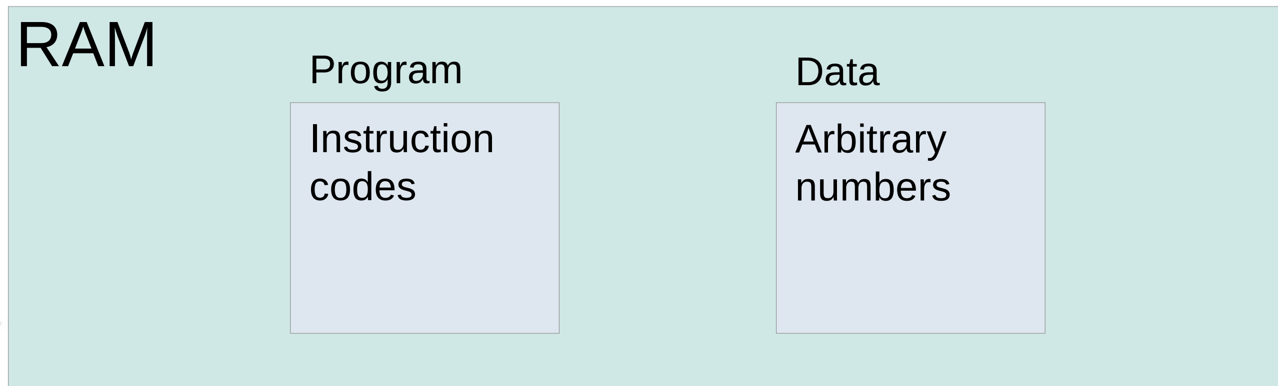
Address	Contents
00000000	01001000
00000001	01100101
00000010	01101100
00000011	01101100
00000100	01101111
00000101	00100000
00000110	01110111

OK:

NTNU – Trondheim
Norwegian University of
Science and Technology

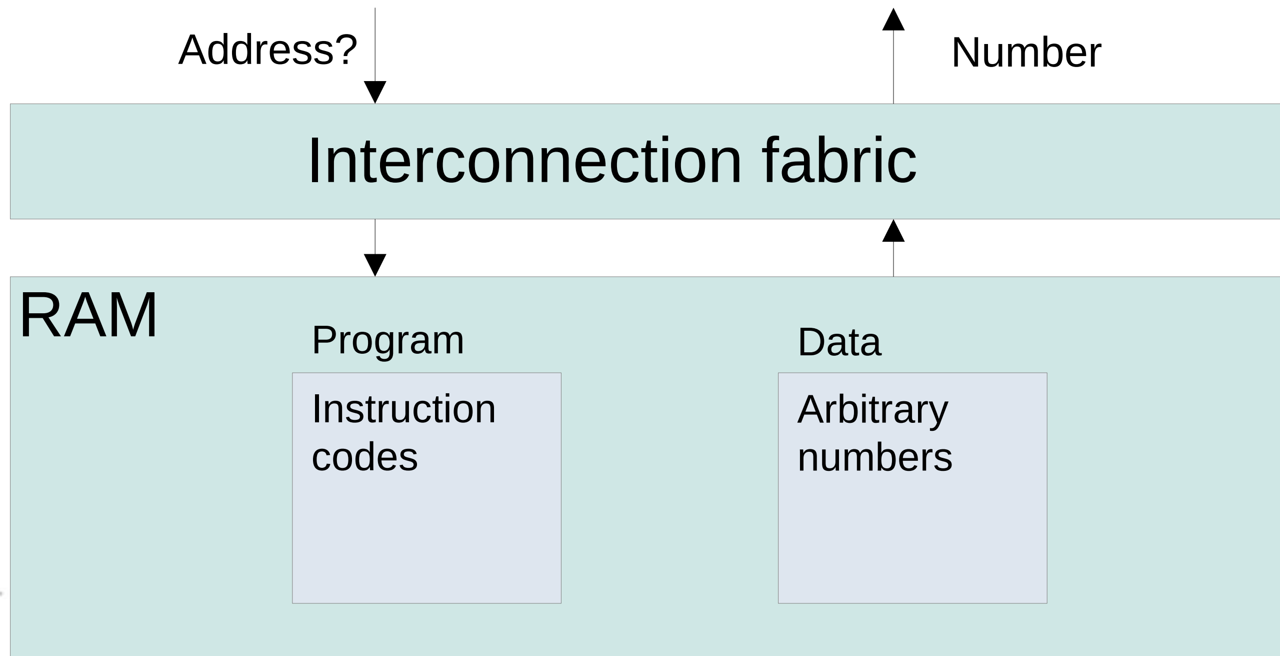
This table is enormous

- We can assign separate parts of it to contain
  - Numbers that represent specific actions (the program)
  - Numbers that we want to apply the actions to (the data)
- They're all just some numbers, but we can feed them into different parts of the processor when we know which numbers are kept where:



We need some wires

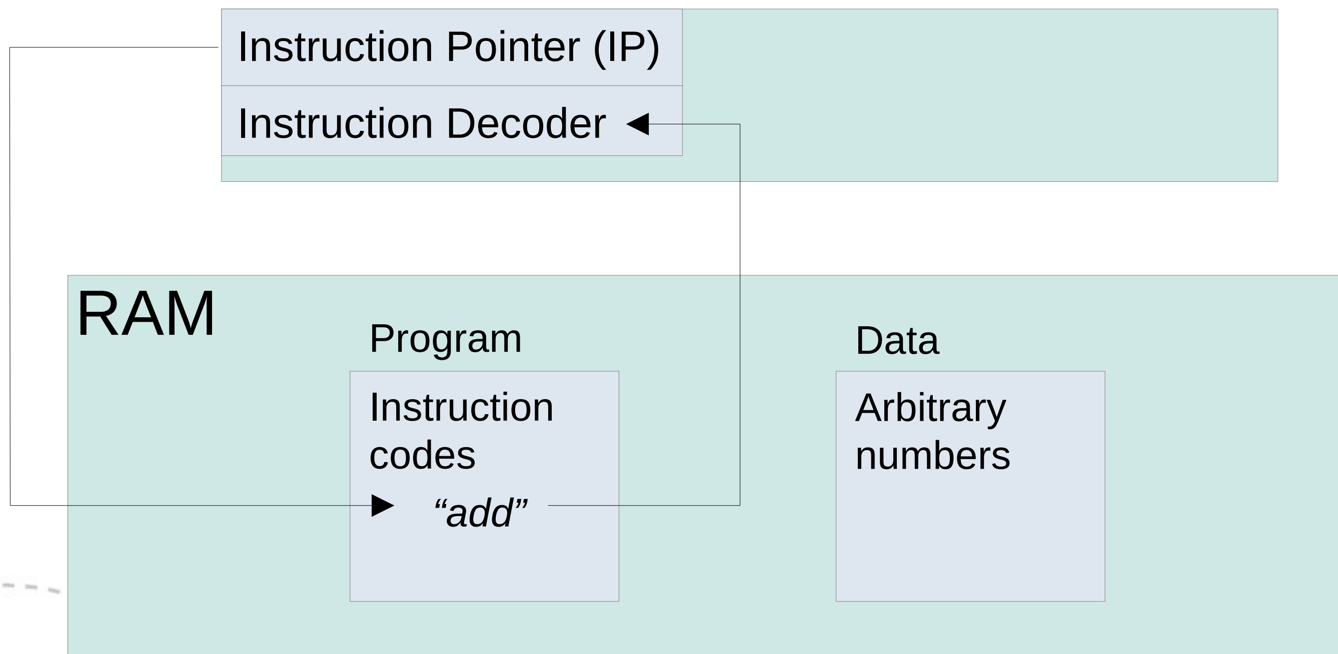
- The interconnect is an interface that lets external devices get numbers from memory
- I'll take it for granted in the next few slides, but it's there.



NTNU – Trondheim
Norwegian University of Science and Technology

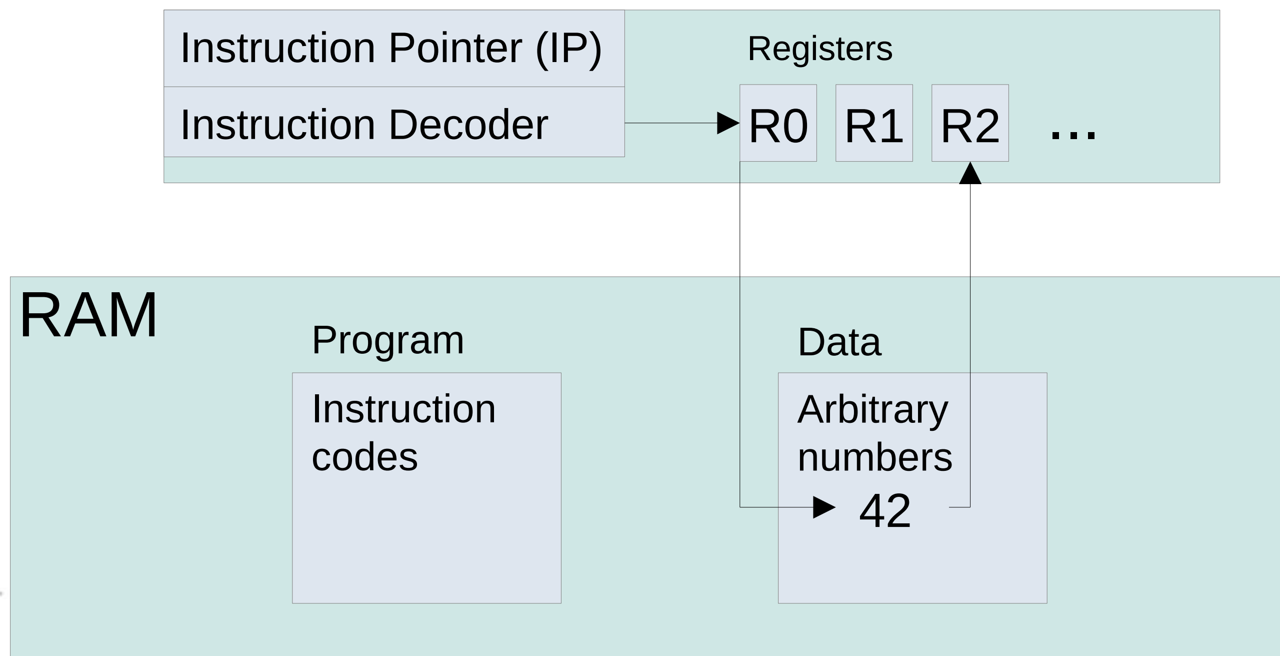
Time to read instructions

- Programs execute one step at a time
  - IP contains the location of the current operation
  - The *control path* feeds it into a gizmo that knows what all the codes mean
  - The IP is updated (mostly, incremented) to point at the next operation when the operation is finished



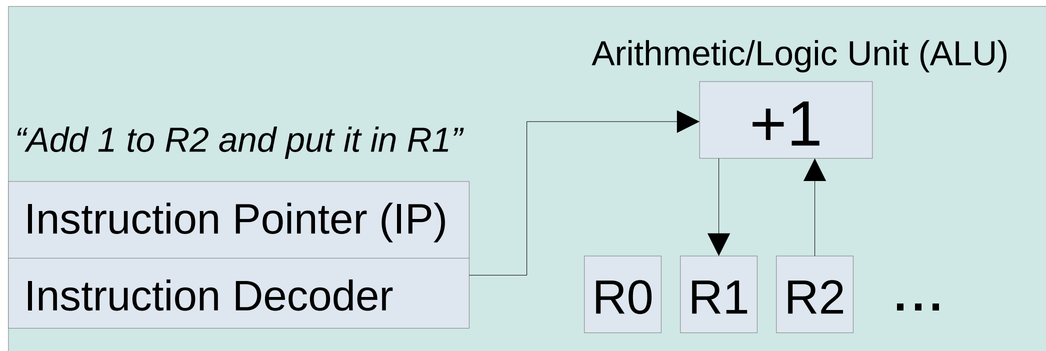
Instructions require data

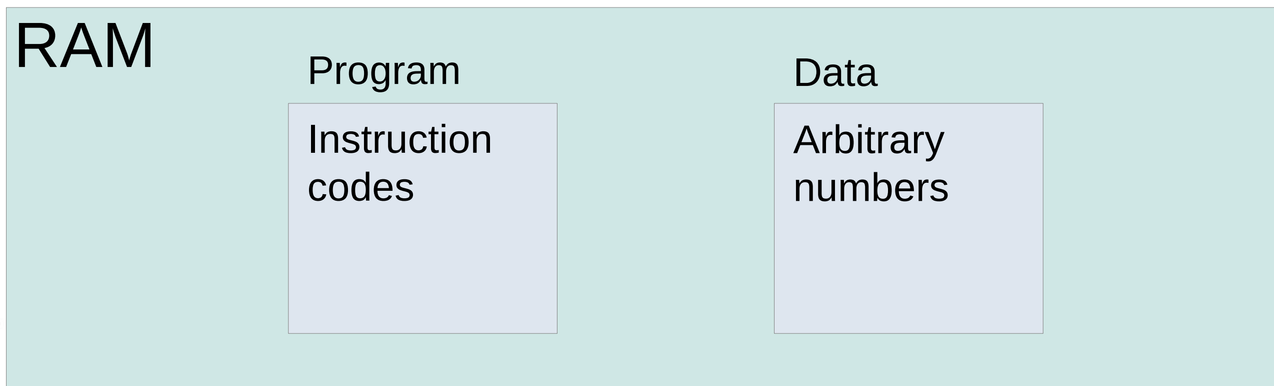
- Some of the instructions just transport data between memory and local storage registers:



Instructions require data

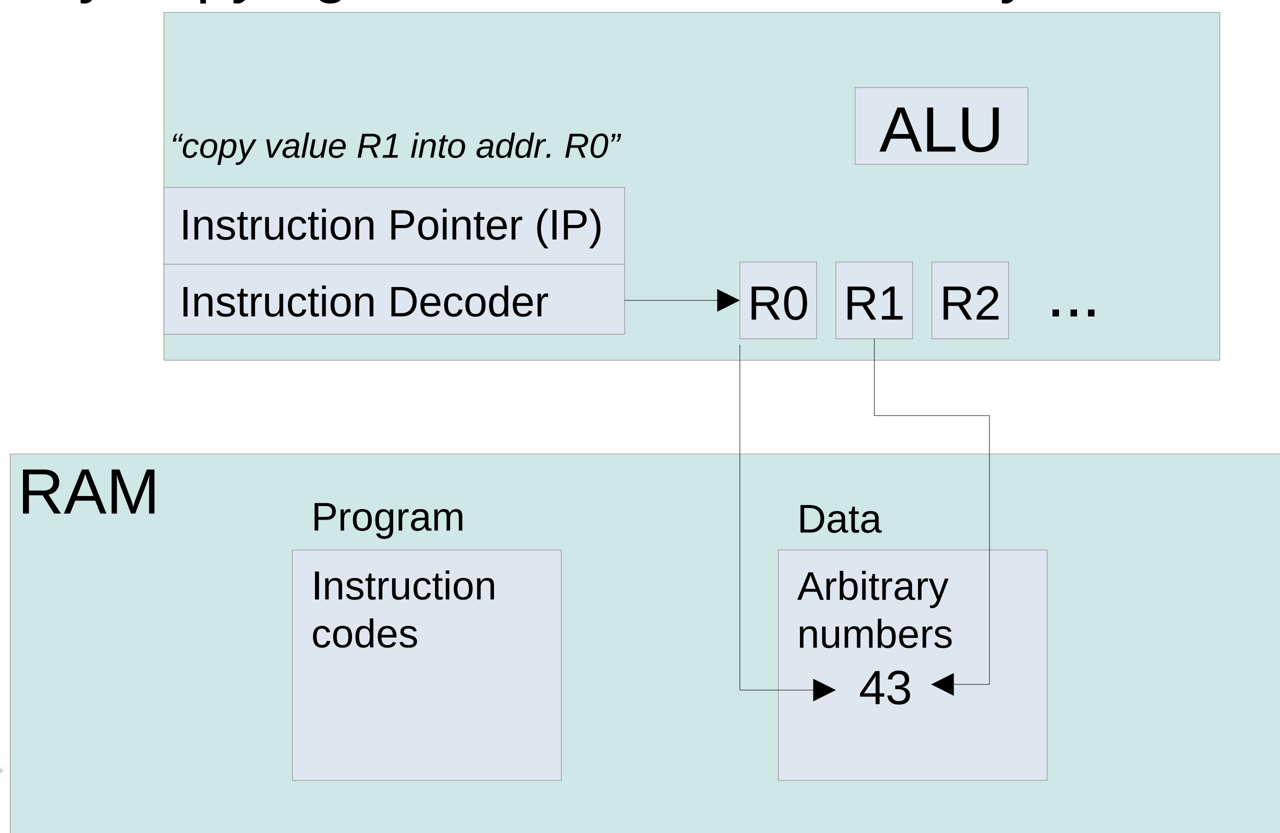
- Other instructions do something to the data in the registers:





Instructions require data

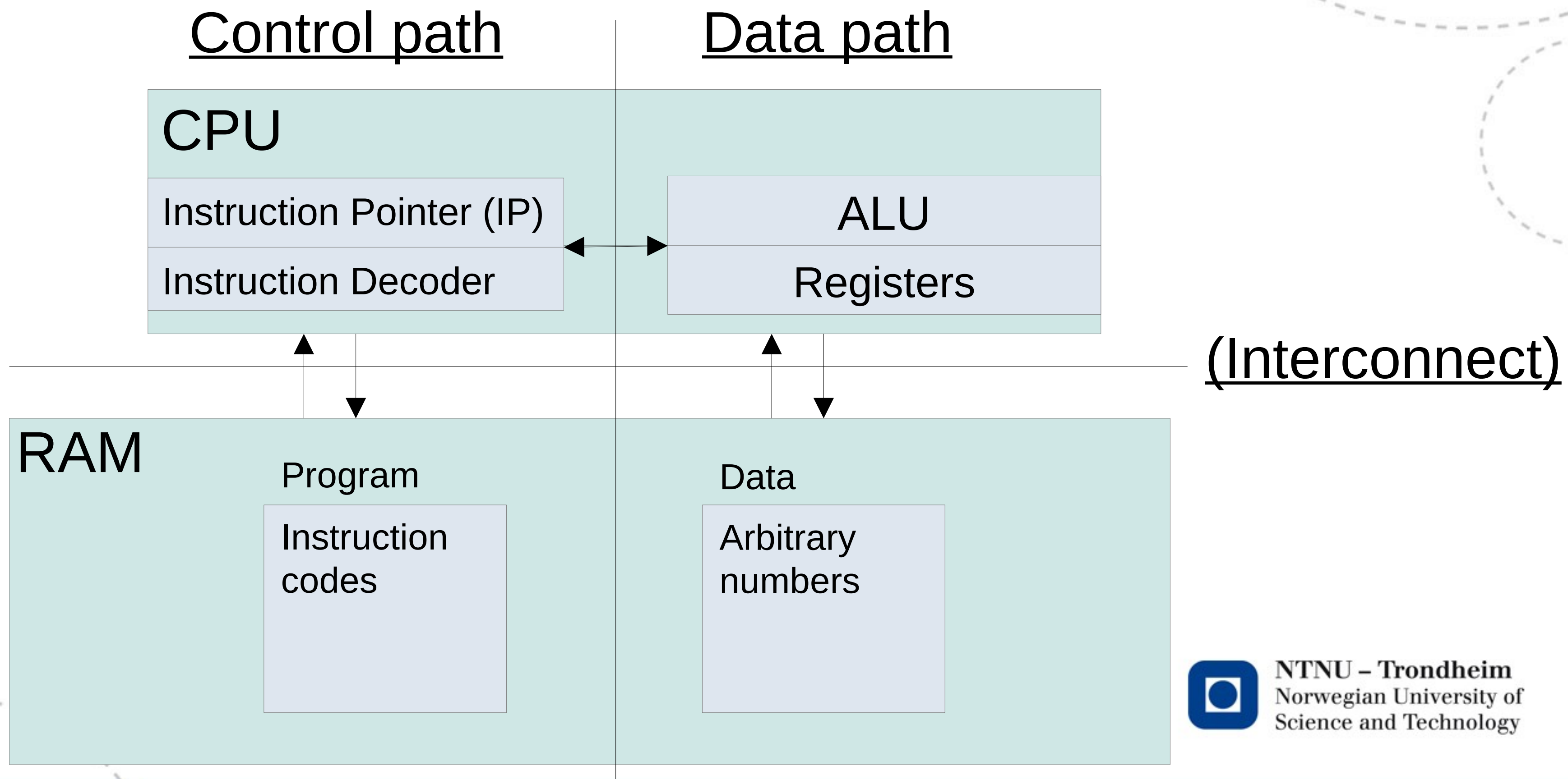
- Temporary results are made (semi-)permanent by copying them back into memory:



NTNU – Trondheim
Norwegian University of Science and Technology

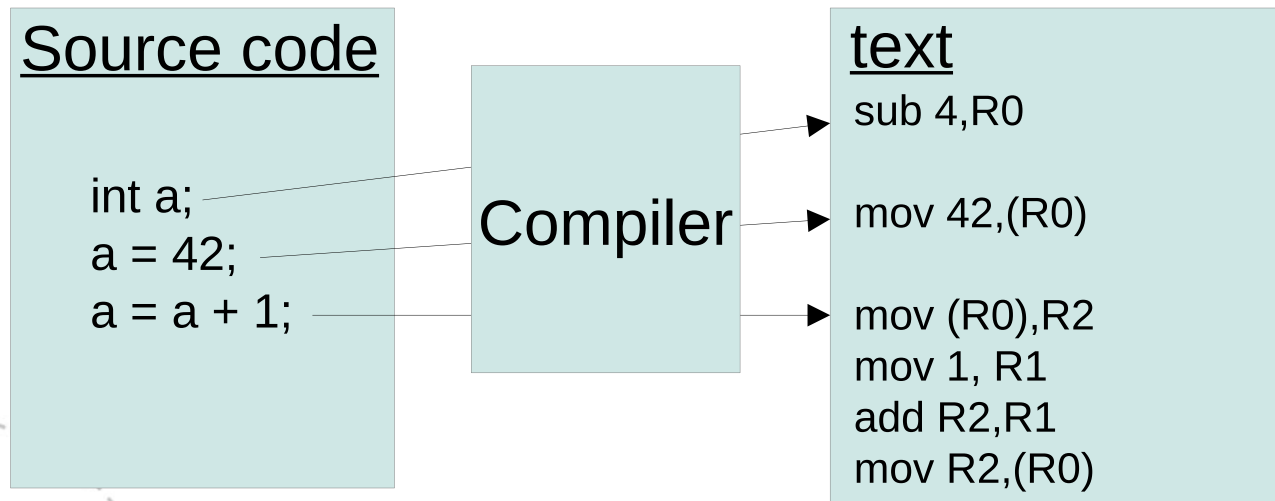
The *von Neumann architecture*

- We now have a (simplified) picture of it:



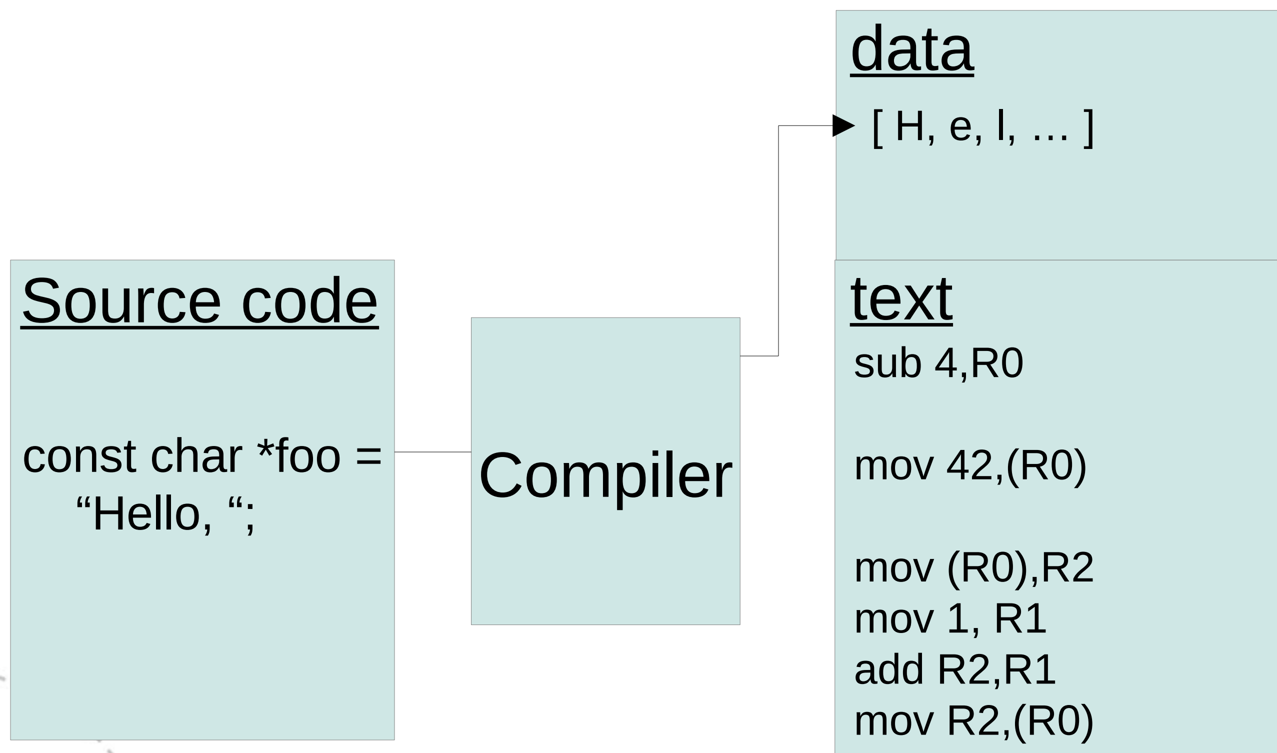
How do we write the instructions?

- They are derived from a more readable text written in some suitable programming language
- *TDT4205 Compiler Construction* deals with how this translation takes place



How do we write the data?

- Constants and initialized global values are also derived from the source program



The executable file

- These are the contents of the binary executable that your compiler produces

<u>data</u>

[H, e, l, …]

<u>text</u>

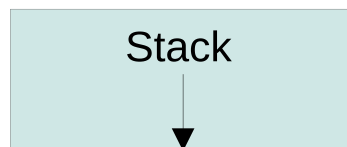
sub 4,R0

mov 42,(R0)

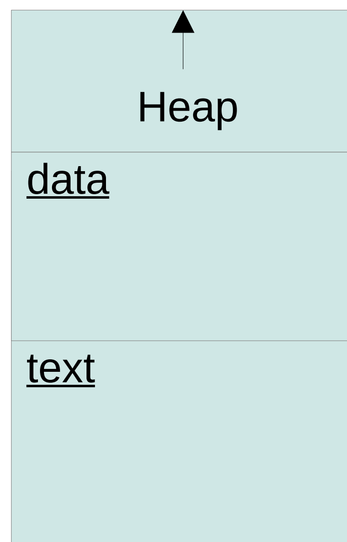
mov (R0),R2
mov 1, R1
add R2,R1
mov R2,(R0)

Intermediate values

- We also need space for data that are produced (and deleted) during program execution:



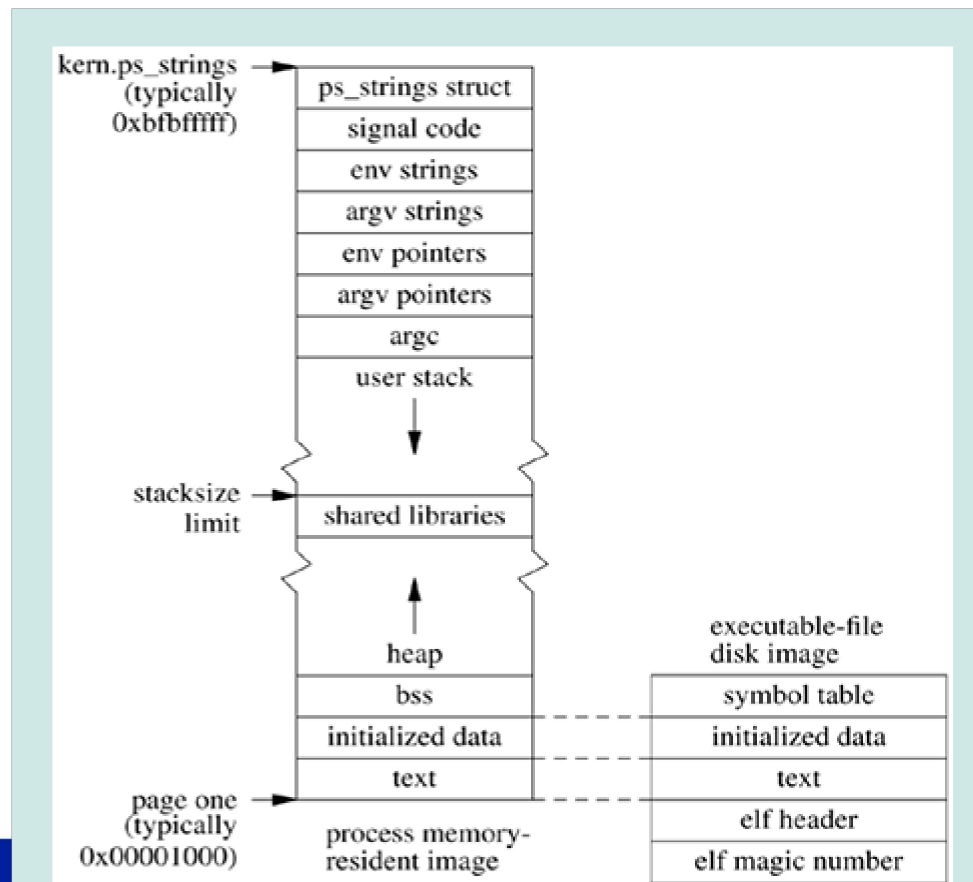
Compiler places function arguments and local variables here



Program code places data of dynamically calculated sizes here

Here's a complete *process image*

- This one is taken from *McKusick & Neville-Neil*'s book about the design of the FreeBSD operating system, but your favorite OS will feature something very similar



Our sequential processing model

- We have an abstract computer, complete with
  - A program to run
  - A translation mechanism that separates data and instructions
  - An instruction-interpreting unit
  - A data-manipulating unit
- It's a *model* because it's a simplified way to reason about what actually happens, which is consistent with the results we obtain
- It's *sequential* because it requires that each instruction is completed before we start work on the next one

The strength of Neumann

- When you're a programmer, you have a simple way to improve system performance:
  - Rewrite the same calculation using *fewer* or *shorter* instructions
- When you're a hardware designer, you have a simple way to improve system performance:
  - Implement the same instructions in ways that work faster
- The von Neumann computer is a *bridging model:*
  - Programmers can improve performance for every computer
  - Hardware designers can improve performance for every program

The weakness(es) of Neumann #1

- Programs become strings of *read-modify-write cycles*
- We illustrated one before, it goes
  - Read data to work on from memory into registers
  - Combine the values in the registers
  - Write the results from registers back into memory
- This means that the program will run at the speed of memory access whenever it needs to load new data
  - ...but in modern computers, the processor can work hundreds of times faster than memory…
- This constraint is known as the *von Neumann bottleneck*

The weakness(es) of Neumann #2

- Sequential programs can only finish as quickly as the sum of their operations
- CPU clock speeds grew rapidly from 1965 until around 2003, but they have leveled off since then
- The only other way to get through all the instructions faster is to work on more than 1 instruction simultaneously
- The von Neumann machine can't do it

The weakness(es) of Neumann #3

- It gives all available memory to every program, and makes no distinction between frequently used and entirely idle addresses
- We can only run 1 program at a time
- It can only use 1 gigantic table of addresses

The bridging model for parallel computing

- We don’t have one.
- Sorry.
- That doesn’t mean that no such thing can exist, but it certainly hasn’t been invented yet
- It is a work in progress
- It has been a work in progress for decades

I strongly advise you not to hold your breath waiting for one

The fallout

- We write programs in the von Neumann style, and try to detect the parts that can be done in parallel
- Some mechanisms find them automatically
- Other mechanisms require them to be explicitly identified by the programmer
- Whether the improvements are automatic or manual, programmers can only improve system performance by adapting their code to the type of computer it targets

(specifically, how its design extends the von Neumann model)

That is our main topic

- We will discuss explicit ways to run
  - Multiple collaborating processes (distributed memory)
  - Multiple instruction streams in one process (shared memory)
  - Multiple operations in one instruction (vector operations)
  - Multiple processor types in one program (hybrid programming)
- …but we'll start with some invisible adaptations in modern systems
  - Cache memory
  - Virtual memory
  - Instruction-Level Parallelism (ILP)
- Stay tuned. :)