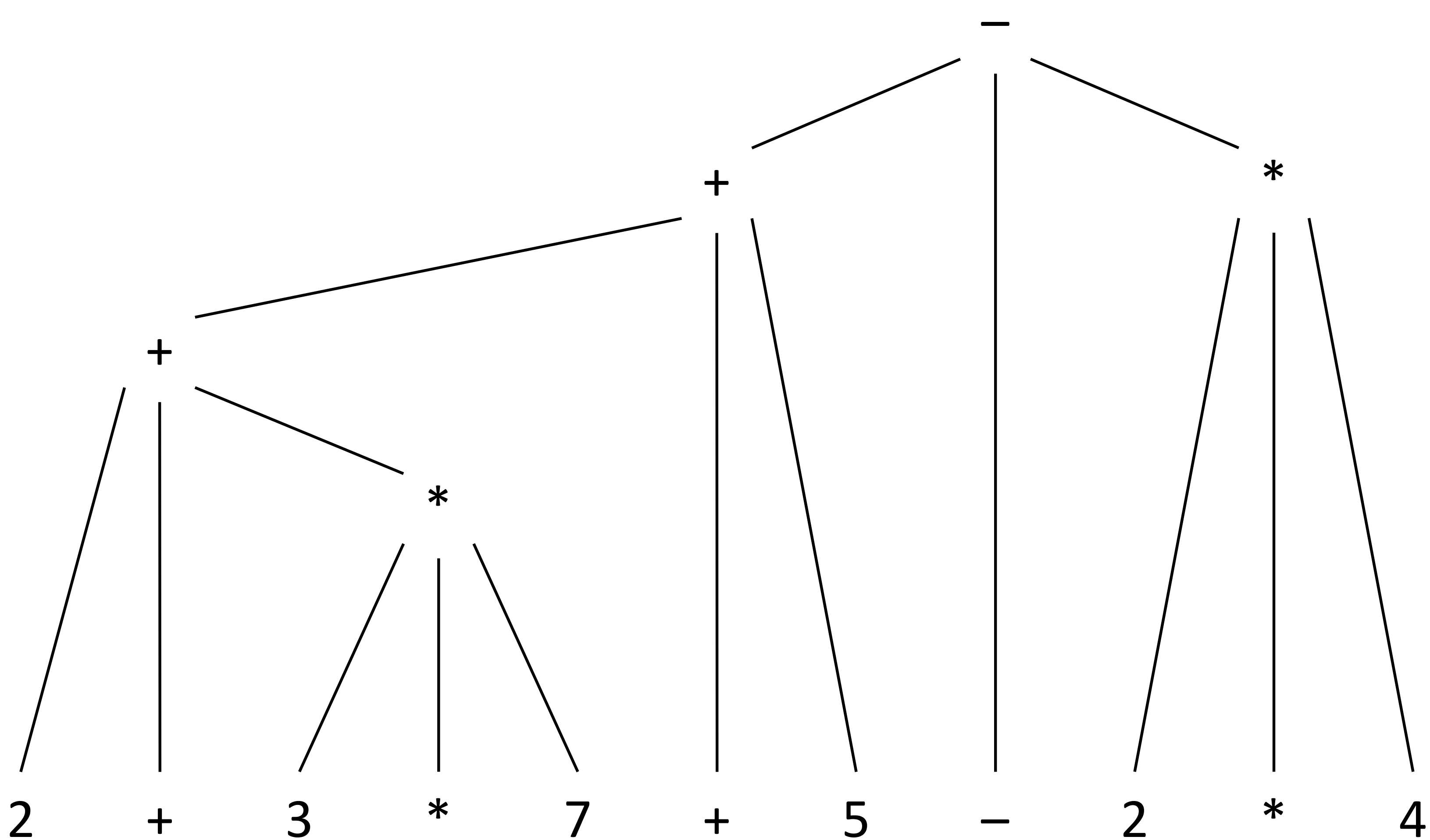
Hardware

Lembit Jürimägi

Software interpreter

- Parser lets us create an Abstract Syntax Tree, that represents the original code in a tree structure
- We can create nodes for arithmetic/logic operations, conditions, branch and loop statements, etc
- Crawling thru the tree structure can be easily done with a recursive function (implicit use of C call stack) or an explicit stack
- However, we are wasting additional time for navigating the tree structure
- This could be rewritten into a linear list of operations
- But we need 2 things, storage and a way to skip operations
- Let's look at some examples

Abstract syntax tree, arithmetic

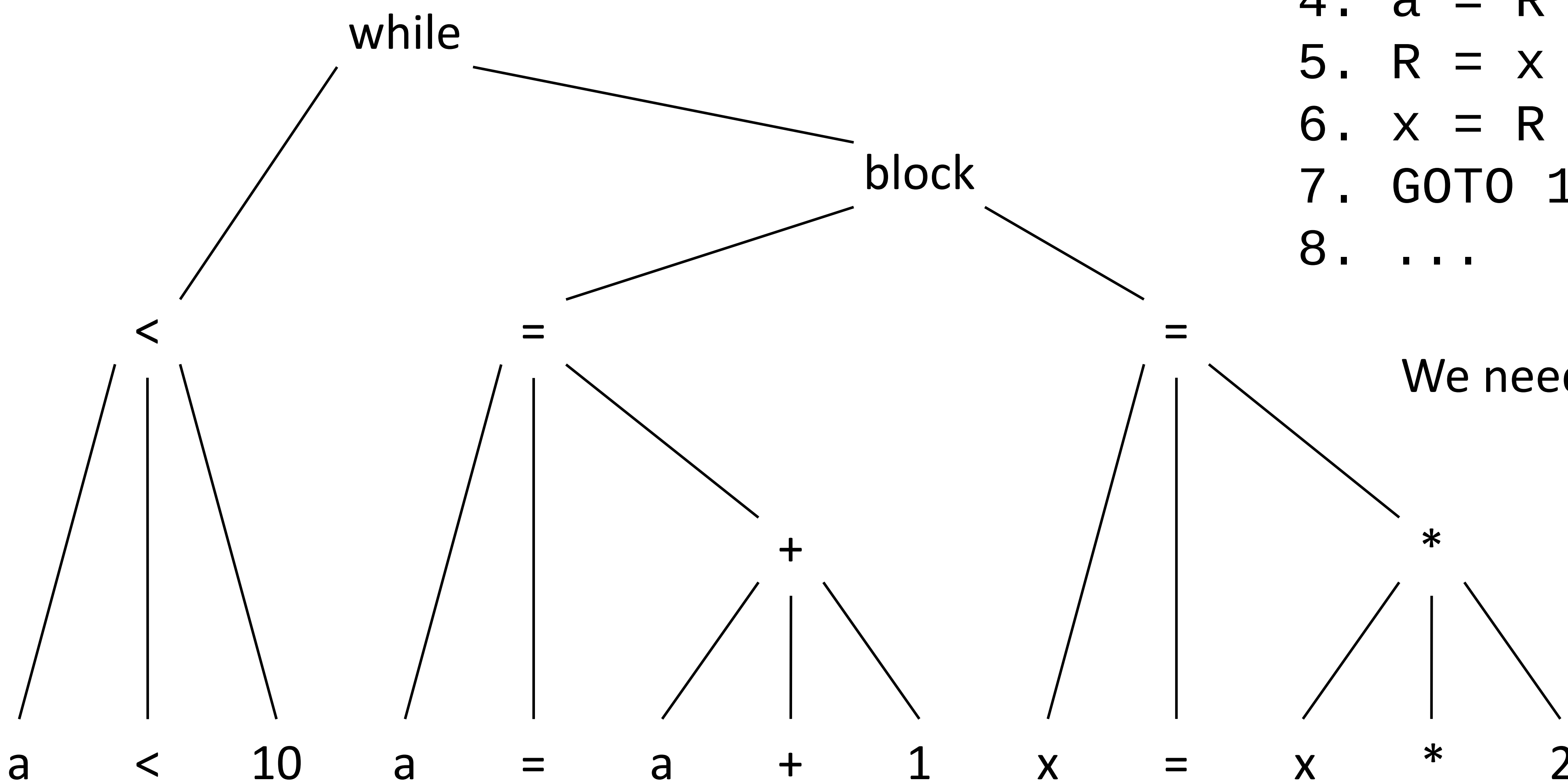


R = 3 * 7
R = 2 + R
R = R + 5
S = 2 * 4
R = R − S

We need storage for R and S

Abstract syntax tree, loop

```
1. R = a – 10
2. ifpos R GOTO 8
3. R = a + 1
4. a = R
5. R = x * 2
6. x = R
7. GOTO 1
8. ...
```



We need GOTO

Hardware

- Hardware could be physical or virtual (Virtual Machine)
- Virtual machine could replicate physical hardware but not always the case
- Instead of statements we have instructions
- High level statements have to be mapped onto one or more instructions

Code vs HW instructions

- Arithmetic and logic operations map very well to instructions
- Loops and conditional statements are replaced by jump/branch
- Conditions are turned into operations (`a < b` becomes `a – b`) or specific branch instructions
- There could be hardware flags that keep track of instruction results (is the result negative, is it zero, was there an overflow, etc)

Hardware design

- We need CPU, storage for program and data, and maybe external devices
- In the CPU we need
  - instruction pointer / program counter
  - a way to access external memory
  - instruction decoder
  - execution unit (ALU)
  - maybe a way to access external devices
  - maybe interrupts
  - maybe FPU
  - maybe internal memory (register file or stack)

CPU internal memory

- Could be small register file (typical to CISC architectures like x86)
- Could be large register file (typical to RISC)
- Could be stack (typical to virtual-only architecture)
- With register file it becomes necessary to map operands to registers
- It is an optimization problem, running out of registers means that operands have to be temporarily moved to memory
- Operands have different lifetime and same register can be reused by variables without overlapping lifetimes

External devices

- External devices (such as display, keyboard, etc) are accessible thru ports
- They may generate interrupts (a key is pressed), or they could be polled (a switch is on) or they could be write-only
- Ports may be mapped to memory (some memory addresses are reserved) or they could have separate address space and separate instructions for dealing with ports
- An interrupt requires a handler (the pressed key is stored in a memory buffer), the processor stops the current program flow, switches contexts, deals with interrupt, returns

Function calls

- Jumping to the subroutine is pretty similar to loops, conditions, etc
- Returning from the subroutine is different, CPU must know where it must return to
- Therefore there are usually separate instructions for calling a function
- Function body has instructions that will alter the contents of registers (if there are registers)
- This could completely mess up the code that called the function
- Calling convention is a standard on what steps are taken by caller and or callee to save the current context and restore it once the call is done

Function call stack

- A function is a separate block of code with its own variables (and parameters)
- A recursive function needs a new set of variables for each iteration
- Best way is to dedicate a separate stack frame for each function that is called
- But that requires hardware support, there must be dedicated registers that keep track of the current frame and let CPU address variables within that frame
- Usually two registers, stack pointer and base pointer
- Older hardware didn't have that, meaning that both parameters and internal variables were global variables

Architectures

- CISC
  - complex instructions
  - few registers
  - instructions can address memory directly
  - variable length instructions
- RISC
  - simple instructions
  - many registers
  - instructions performed with registers and immediate values
  - instructions are fixed length
- CISC-RISC
  - complex instructions
  - mapped to internal smaller instructions (μops)
  - take variable amount of clock cycles

Data types

- Programming language can have several datatypes
- For example, in C, 8 integer types (1, 2, 4, 8 bytes) x (signed, unsigned)
- Then 3 floating point types (float, double, long double)
- An arithmetic operation can be performed between any of these types ((11*10)/2 combinations)
- When mapping these data types to native hardware, additional conversion steps may be necessary

Java Virtual Machine

- Java code is compiled to byte-code
- The local data is in stack
- Instructions are single byte, operands and result are in the stack
- JVM emulates a theoretical 32-bit machine
- Emulation is slow, so there is also just-in-time compiler to compile byte-code to native physical code
- .NET has similarities