System software

Resolving symbols

Code representation

- Class `Code`
  - name, start address, program
  - program is an array of Nodes
    - `List<Node> program`
  - symbol table
    - `Map<String, Integer> symbols`
    - `defineSymbol(sym, val)`
    - `resolveSymbol(sym)`

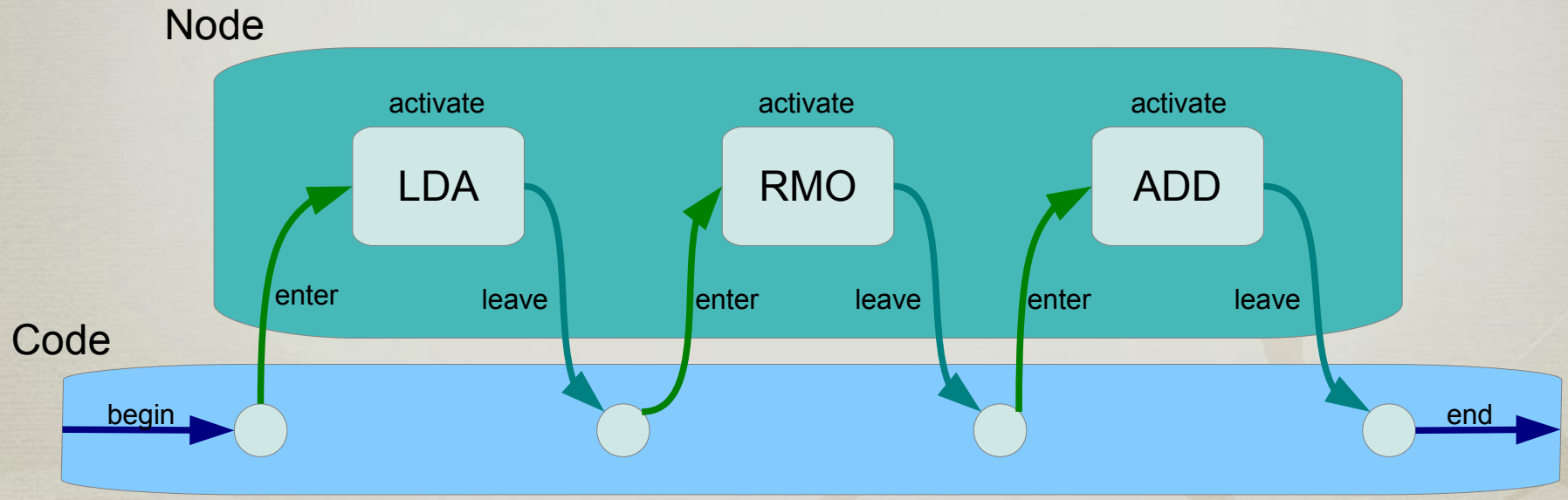
- Node
  - Comment
  - InstructionF1
  - InstructionF2
  - InstructionF3
  - InstructionF4
  - Directive
  - Storage

Code representation

- Class `Node`.
  - `String label`
  - `Mnemonic mnemonic`
  - `String comment`
  - `toString()`
  - `…`

Code visitation

- Visitor design pattern (simplified)
  - commands are visited sequentially



Code visitation

- Class `Code`
  - `begin()` … start of traversal
    - initialization of LOCCTR (location counter)
      - `loc = start; nextLoc = start`
    - initialization of base addressing
      - `regB = -1`
  - `end()` … end of traversal
    - any extensions

Code visitation

- Class `Code`
  - visitors
    - `resolve()`
      - resolving symbols
    - `dumpText()`
      - generating object file
    - `dumpCode()`
      - generating raw code
    - etc

```
public void resolve() throws SemanticError {
      begin();
      for (Node node : program) {
            node.enter(this);
            node.resolve(this);
            node.leave(this);
      }
      length = nextLoc - start;
      end();
}
```

Code visitation

- Class `Node`
  - `enter(Code code)` ... command enter
    - `code.loc = code.nextLoc;`
    - `code.nextLoc += length();`
  - `leave(Code code)` ... command leave
    - directive ORG

Code visitation

- Class `Node`
  - different visitor
    - `activate(Code code)`
      - *1st pass (see lectures)*
      - defines symbol (label) in the symbol table
    - `resolve(Code code)`
      - *2nd pass (see lectures)*
      - resolve symbols in formats F3, F4
    - `emitCode(byte[] data, int pos)`
    - `emitText(StringBuffer buf)`
      - use `emitCode()` and transform data into buf
      - but be careful with `RESB` and `RESW`

First pass

- Reading and parsing the source code
  - adding the command to the AST
- Filling up the symbol table
  - all labels (left symbols) are defined
- Visitation
  - code.append(Node node)
    - program: add(node)
    - node: enter() activate(), leave()

Second pass

- Resolving right symbols
  - based on the symbol table
  - replace right symbols with addresses
- Resolving addressing
  - address use
    - immediate, simple, indirect
  - address calculation
    - PC-relative, base-relative, direct (absolute)

Address use

- Bits `ni` and `x`.
  - can be treated already in the first pass
  - `x` – indexed addressing

`n`	`i`	operand	description
0	0	(addr)	simple – SIC format
0	1	addr	immediate (slov. *takojšnje*)
1	0	((addr))	indirect (slov. *posredno*)
1	1	(addr)	simple (slov. *preprosto*)

Address calculation

- Bits bp.
  - SIC/XE format 3
  - the bits are determined when resolving symbols

b	p	calculation	description
0	0	disp	direct
0	1	(PC) + disp	PC-relative (2048 <= disp <= 2047)
1	0	(B) + disp	B-relative (0 <= disp <= 4095)
1	1		invalid / undefined

Resolving F3

- Try PC-relative
  - -2048 ≤ displacement from the PC register ≤ 2047
- Try base-relative
  - 0 ≤ displacement from the B register ≤ 4095
- Try direct (absolute)
  - 0 ≤ address ≤ 4095
  - relocatable code?
- * Try SIC format, direct (absolute)
  - 0 ≤ address ≤ 32767 (15 bits)