A history of (Nordic) compilers and autocodes

Peter Sestoft
sestoft@itu.dk

2014-10-13
Copenhagen Tech Polyglot Meetup
The speaker

- MSc 1988 computer science and mathematics and PhD 1991, DIKU, Copenhagen University
- KU, DTU, KVL and ITU; and AT&T Bell Labs, Microsoft Research UK, Harvard University
- Programming languages, software development, ...
- Open source software
  - Moscow ML implementation, 1994...
  - C5 Generic Collection Library, with Niels Kokholm, 2006...
  - Funcalc spreadsheet implementation, 2014
# Practical Concurrent and Parallel Programming (PCPP) (PRCPP)

- This MSc course is about how to write correct and efficient concurrent and parallel software, primarily using Java, on standard shared-memory multicore hardware. It covers basic mechanisms such as threads, locks and shared memory as well as more advanced mechanisms such as transactional memory, message passing, and compare-and-swap. It covers concepts such as atomicity, safety, liveness and deadlock. It covers how to measure and understand performance and scalability of parallel programs. It covers tools and methods find bugs in concurrent programs.
- For exercises, quizzes, and much more information, see the course LearnIT site (restricted access).
- For formal rules, see the official course description.

## Lecture plan

<table>
<thead>
<tr>
<th>Course week</th>
<th>ISO week</th>
<th>Date</th>
<th>Who</th>
<th>Subject</th>
<th>Materials</th>
<th>Exercises</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
<td>29 Aug</td>
<td>PS</td>
<td>Concurrent and parallel programming, why, what is so hard. Threads and locks in Java, shared mutable memory, mutual exclusion, Java inner classes.</td>
<td>Goetz chapters 1, 2; Sutter paper; McKenney chapter 2; Bloch item 66; Slides week 1; Exercises week 1; Example code: pcpp-week01.zip</td>
<td>Exercises week 1</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>5 Sep</td>
<td>PS</td>
<td>Threads and Locks: Threads for performance, sharing objects, visibility, volatile fields, atomic operations, avoiding sharing (thread confinement, stack confinement), immutability, final, safe publication</td>
<td>Goetz chapters 2, 3; Bloch item 15; Slides week 2; Mandatory exercises week 2; Example code: pcpp-week02.zip</td>
<td>Mandatory handin 1</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
<td>12 Sep</td>
<td>PS</td>
<td>Threads and Locks: Designing thread-safe classes. Monitor pattern. Concurrent collections. Documenting thread-safety.</td>
<td>Goetz chapters 4, 5; Slides week 3; Exercises week 3; Example code: pcpp-week03.zip</td>
<td>Exercises week 3</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>19 Sep</td>
<td>PS</td>
<td>Performance measurements.</td>
<td>Sestoft: Microbenchmarks; Slides week 4; Exercises week 4; Example code: pcpp-week04.zip; Optional: McKenney chapter 3</td>
<td>Mandatory handin 2</td>
</tr>
<tr>
<td>5</td>
<td>39</td>
<td>26 Sep</td>
<td>PS</td>
<td>Threads and Locks: Tasks and the Java executor framework. Concurrent pipelines, wait() and notifyAll().</td>
<td>Goetz chapters 6, 8; Bloch items 68, 69; Slides week 5; Exercises week 5; Example code: pcpp-week05.zip</td>
<td>Exercises week 5</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>3 Oct</td>
<td>PS</td>
<td>Threads and Locks: Safety and liveness, absence of deadlock and livelock. The ThreadSafe tool.</td>
<td>Goetz chapter 10, 13.1; Bloch item 67; Slides week 6; Exercises week 6; Example code: pcpp-week06.zip</td>
<td>Mandatory handin 3</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>10 Oct</td>
<td>PS</td>
<td>Threads and Locks: Performance and scalability</td>
<td>Goetz chapter 11, 13.5; Slides week 7; Exercises week 7; Example code: pcpp-week07.zip</td>
<td>Exercises week 7</td>
</tr>
</tbody>
</table>

http://www.itu.dk/people/sestoft/itu/PCPP/E2014/
The future is parallel – and functional

- Classic imperative for-loop to count primes:

  ```java
  int count = 0;
  for (int i=0; i<range; i++)
      if (isPrime(i))
          count++;
  ```

- Sequential functional Java 8 stream:

  ```java
  IntStream.range(0, range)
      .filter(i -> isPrime(i))
      .count();
  ```

- Parallel functional stream:

  ```java
  IntStream.range(0, range)
      .parallel()
      .filter(i -> isPrime(i))
      .count();
  ```

  - i7: 9.9 ms
    AMD: 40.5 ms
  - i7: 9.9 ms
    AMD: 40.8 ms
  - i7: 2.8 ms
    AMD: 1.7 ms
  - i7: 3.6 x speedup
    AMD: 24.2 x speedup
    for free
Outline

• What is a compiler?
• Genealogies of languages and of early computers
• Knuth's survey of early autoprogramming systems
• Lexing and parsing
• Compilation of expressions
• FORTRAN I in the USA
• Algol 60 in Europe
• Early Nordic autocodes and compilers
• (Intermediate languages)
• (Optimization)
• (Flow analysis)
• (Type systems)
• (Compiler generators)
• The nuclear roots of object-oriented programming
What is a compiler? and autocode?

C language source program

```c
for (int i=0; i<n; i++)
    sum += sqrt(arr[i]);
```

clang

x86 machine code

```
LBB0_1:
    movl  -28(%rbp), %eax     // i
    movl  -4(%rbp), %ecx      // n
    cmpl  %ecx, %eax
    jge   LBB0_4              // if i >= n, return
    movslq -28(%rbp), %rax   // i
    movq   -16(%rbp), %rcx   // address of arr[0]
    movsd (%rcx,%rax,8), %xmm0 // arr[i]
    callq _sqrt                // sqrt
    movsd -24(%rbp), %xmm1    // sum
    addsd %xmm0, %xmm1         // sum + ...
    movsd %xmm1, -24(%rbp)     // sum = ...
    movl -28(%rbp), %eax      // i
    addl $1, %eax              // i + 1
    movl %eax, -28(%rbp)       // i = ...
    jmp LBB0_1                 // loop again
```

autocode (early compilers)

From Aho et al.
Conceptual phases of a compiler
Genealogy of programming languages
Stored program computers

• Programs and data stored in the same way
  – EDVAC and IAS designs ("von Neumann") 1945
• So: program = data
• So a program can process another program
  – This is what a compiler or assembler does

• Also, a program can modify itself at runtime
  – Used for array indexing in IAS, EDSAC, BESK, ...
  – Used for subroutine return, EDSAC, the "Wheeler jump"

• Modern machines use index registers
  – For both array indexing and return jumps
  – Invented in Manchester Mark I, 1949
  – Adopted in the Copenhagen DASK 1958
A history of the history of ...

  - Many references to important early papers
  - USSR addendum by Ershov in 2nd printing (1976)
  - Opening quote:

  D. E. KNUTH [81] has observed (in 1962!) that the early history of compiler construction is difficult to assess. Maybe this, or maybe the general unhistorical attitude of our century is responsible for the widespread ignorance about the origins of compiler construction. In addition, the overwhelming lead of the USA in the general de-
Some older histories of ...

- Knuth: A *history of writing compilers* (1962)
  - Few references, names and dates, mostly US:

  > A complete bibliography of the compiler literature is hard to give; you may, in fact, find it quite distressing to try to read many of the articles.

  - Also lists people interested in automatic coding
  - Only US and UK: Cambridge and Manchester
# Knuth 1977: The early development ...

<table>
<thead>
<tr>
<th>Language</th>
<th>Principal author(s)</th>
<th>Year</th>
<th>Arithmetic</th>
<th>Implementation</th>
<th>Readability</th>
<th>Control structures</th>
<th>Data structures</th>
<th>Machine independence</th>
<th>Impact</th>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plankalkül</td>
<td>Zuse</td>
<td>1945</td>
<td>X, S, F</td>
<td>F</td>
<td>D</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>Programming language, hierarchic data</td>
</tr>
<tr>
<td>Flow diagrams</td>
<td>Goldstine &amp; von Neumann</td>
<td>1946</td>
<td>X, S</td>
<td>F</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>Acceptable data handling</td>
</tr>
<tr>
<td>Composition</td>
<td>Curry</td>
<td>1948</td>
<td>X</td>
<td>F</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>Algorithm</td>
</tr>
<tr>
<td>Short Code</td>
<td>Mauchly</td>
<td>1950</td>
<td>F</td>
<td>C</td>
<td>C</td>
<td>F</td>
<td>C</td>
<td>F</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Intermediate PL</td>
<td>Burks</td>
<td>1950</td>
<td>?</td>
<td>F</td>
<td>A</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Klammer-ausdrücke</td>
<td>Rutishauser</td>
<td>1951</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formules</td>
<td>Böhm</td>
<td>1951</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUTOCODE</td>
<td>Glennie</td>
<td>1951</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-2</td>
<td>Hopper</td>
<td>1951</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Whirlwind translator</td>
<td>Lanning &amp; Zimet</td>
<td>1951</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUTOCODE</td>
<td></td>
<td>1952</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIII</td>
<td></td>
<td>1955</td>
<td>F</td>
<td>B</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>Book about a compiler</td>
</tr>
<tr>
<td>BACAIC</td>
<td></td>
<td>1955</td>
<td>F</td>
<td>A</td>
<td>A</td>
<td>D</td>
<td>F</td>
<td>A</td>
<td>D</td>
<td>Use on two machines</td>
</tr>
<tr>
<td>Kompiler 2</td>
<td></td>
<td>1955</td>
<td>S</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>Scaling aids</td>
</tr>
<tr>
<td>PACT I</td>
<td></td>
<td>1955</td>
<td>X, S</td>
<td>A</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>Cooperative effort</td>
</tr>
<tr>
<td>ADES</td>
<td></td>
<td>1956</td>
<td>X, F</td>
<td>D</td>
<td>D</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>F</td>
<td>Declarative language</td>
</tr>
<tr>
<td>IT</td>
<td></td>
<td>1956</td>
<td>X, F</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>Successful compiler</td>
</tr>
<tr>
<td>FORTRAN I</td>
<td></td>
<td>1956</td>
<td>X, F</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>I/O formats, comments, global optimization</td>
</tr>
<tr>
<td>MATH-MATIC</td>
<td>Katz</td>
<td>1956</td>
<td>F</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>A</td>
<td>Heavy use of English</td>
</tr>
<tr>
<td>Patent 3,047,228</td>
<td>Bauer &amp; Samelson</td>
<td>1957</td>
<td>F</td>
<td>D</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>C</td>
<td>Formula-controlled computer</td>
</tr>
</tbody>
</table>

X=int, F=float, S=scaled  
A ... F = much ... little
Adding the Nordics and Algol, Simula

<table>
<thead>
<tr>
<th>Language</th>
<th>Machine</th>
<th>Operatic</th>
<th>Developer</th>
<th>Comp. size</th>
<th>Comp spec</th>
<th>Citation 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedcode</td>
<td>IBM 701</td>
<td>Sep-1953</td>
<td>Backus</td>
<td></td>
<td></td>
<td>Backus:1954: TheIBM</td>
</tr>
<tr>
<td>A-2</td>
<td>Univac I</td>
<td>Nov-1953</td>
<td>Hopper</td>
<td></td>
<td></td>
<td>Knuth:1977: TheEarly</td>
</tr>
<tr>
<td>Autocode</td>
<td>Mark I</td>
<td>Dec-1955</td>
<td>Brooker</td>
<td></td>
<td></td>
<td>Knuth:1977: TheEarly</td>
</tr>
<tr>
<td>Fortran I</td>
<td>IBM 704</td>
<td>Jun-1957</td>
<td>Backus et al</td>
<td>24000 ins</td>
<td>8 cards/min</td>
<td>Backus:1957: TheFortran</td>
</tr>
<tr>
<td>MAC</td>
<td>Ferranti Mercu</td>
<td>Dec-1957</td>
<td>Dahl</td>
<td></td>
<td></td>
<td>AMS:1958: MathematicalTable</td>
</tr>
<tr>
<td>Runcible</td>
<td>IBM 650</td>
<td>Dec-1958</td>
<td>Knuth</td>
<td></td>
<td></td>
<td>Knuth:1959: Runcible</td>
</tr>
<tr>
<td>Algol 58</td>
<td>Zuse Z22</td>
<td>Dec-1958</td>
<td>Bauer, Samelson</td>
<td></td>
<td></td>
<td>Samelson:1960: SequentialFo</td>
</tr>
<tr>
<td>Algol 58</td>
<td>B 220</td>
<td>Dec-1959</td>
<td>Barton</td>
<td>3500 ins</td>
<td>500 instr/min</td>
<td>Barton:1961: AnotherNameless</td>
</tr>
<tr>
<td>Lisp 1</td>
<td>IBM 704</td>
<td>Jan-1960</td>
<td>McCarthy</td>
<td></td>
<td></td>
<td>McCarthy:1960: RecursiveFun</td>
</tr>
<tr>
<td>Algol 60</td>
<td>X-1</td>
<td>Jun-1960</td>
<td>Dijkstra</td>
<td>2500 words</td>
<td></td>
<td>Dijkstra:1960: RecursiveProgr</td>
</tr>
<tr>
<td>Algol 58</td>
<td>B 205</td>
<td>Sep-1960</td>
<td>Knuth</td>
<td>4000 words</td>
<td>45 cards/min</td>
<td>Knuth:1960: TheInternals</td>
</tr>
<tr>
<td>Algol 60</td>
<td>CDC 1604</td>
<td>Jun-1961</td>
<td>Irons</td>
<td>800 ins/10000 tbl</td>
<td>300 instr/s</td>
<td>Irons:1961: ASyntax</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Facit EDB</td>
<td>Oct-1961</td>
<td>Dahlstrand</td>
<td></td>
<td></td>
<td>Dahlstrand:2009: Minnen</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Gier</td>
<td>Sep-1962</td>
<td>Jensen, Naur</td>
<td>4000 words</td>
<td>30 instr/s</td>
<td>Naur:1963: TheDesign1</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Whet</td>
<td>Sep-1962</td>
<td>Randell, Russel</td>
<td>3000 words</td>
<td>input limited</td>
<td>Randell:1964: Algol60Implement</td>
</tr>
<tr>
<td>Algol 60</td>
<td>Kidsg</td>
<td>Dec-1962</td>
<td>Hawkins, Huxt</td>
<td>20000 words</td>
<td></td>
<td>Randell:1964: Algol60Implement</td>
</tr>
<tr>
<td>Algol 60</td>
<td>M-20</td>
<td>Jan-1964</td>
<td>Ershov</td>
<td>45000 words</td>
<td></td>
<td>Ershov:1966: Alpha</td>
</tr>
</tbody>
</table>
History: lexing and parsing

• Initially ad hoc
• Table-driven/automata methods
• Regular expressions, context-free grammars
• Finite state automata and pushdown automata
• Knuth LR parsing 1965
• Gries operator grammars 1968
• Lexer and parser generator tools
  – Lex (Lesk 1975) and Yacc (Johnson 1975)
  – LR dominated for a while
  – LL back in fashion today: Antlr, Coco/R, parser combinators, packrat parsers

Samelson: 1960: SequentialFormula
Irons: 1961: ASyntax
Naur: 1963: TheDesign1
Knuth: 1965: OnThe
Gries: 1968: UseOf
• Historically, too much emphasis on parsing?
  – Because it was formalizable and respectable?
  – But also beautiful relations to complexity and computability …
History: compilation of expressions

- Rutishauser 1952 (not impl.)
  - Translating arithmetic expressions to 3-addr code
  - Infix operators, precedence, parentheses
  - Repeated scanning and simplification
- Böhm 1952 (not impl.)
  - Single scan expression compilation – also at ETHZ
- Fortran I, 1957
  - Baroque but simple treatment of precedence (Böhm &)
  - Complex, multiple scans
- Samelson and Bauer 1960
  - One scan, using a stack ("cellar") at compile-time
- Floyd 1961
  - One left scan, one right scan, optimized code
Multi-pass gradual compilation of expression

\[ K_1 : \left[ A_1 : ( A_2 + A_3 ) \right] - ( A_1 \times A_2 \times A_3 ) \neq B \]

1. Red.: \( H_1 = 3 \), \( i_1 = 5 \), \( m = 2 \); \( R_1 = A_2 + A_3 \)

\[ K_2 : \left[ A_1 : R_1 \right] - ( A_1 \times A_2 \times A_3 ) \neq B \]

Seems used also by
- First BESM-I Programming Programme, Ershov 1958
Corrado Böhm, ETH Zürich 1951

• An abstract machine, a language, a compiler
  – Three-address code with indirect addressing
  – Machine is realizable in hardware but not built
  – Only assignments \( m + 1 \rightarrow m \); goto \( C \) is: \( C \rightarrow \pi \)
  – Compiler written in the compiled language
  – Single-pass compilation of fully paren. expressions

```plaintext
<table>
<thead>
<tr>
<th>Nature ( r ) du dernier symbole</th>
</tr>
</thead>
<tbody>
<tr>
<td>) ( \rightarrow \pi )</td>
</tr>
<tr>
<td>( ( \rightarrow \pi )</td>
</tr>
<tr>
<td>( \rightarrow \pi )</td>
</tr>
<tr>
<td>( \rightarrow \pi )</td>
</tr>
<tr>
<td>( \rightarrow \pi )</td>
</tr>
<tr>
<td>( \rightarrow \pi )</td>
</tr>
<tr>
<td>var. ( \rightarrow \pi )</td>
</tr>
<tr>
<td>( \rightarrow \pi )</td>
</tr>
</tbody>
</table>

Expression compiler transition table

Implementation of transitions, goto

\[
5 \cdot r + s + t \rightarrow \pi
\]
Bauer and Samelson, Munich 1957: Sequential formula translation

- Using two stacks for single-pass translation
- Takes operator precedence into account
  - so unlike Böhm does not need full parenthetization
Die bekannten Rechenautomaten und Datenverarbeitungsanlagen erfordern im Einzelfall Anweisungen über die Art und den Ablauf der numerischen oder sonstigen informationsverarbeitenden Prozesse. Die Schreibweise, in der diese Anweisungen fixiert werden, wurde zu Beginn der Entwicklung so gewählt, daß sie gewisse als elementar erachtete technische Funktionen der Anlage beschrieb. Die so geschriebenen Anweisungen werden üblicherweise »Programm« genannt. Das Programm für einen Rechennprozeß etwa und die mathematische Formel, mit der der Mathematiker diesen Prozeß gewöhnlich beschreibt, müssen diesen derselben Vor-
History: Compilation techniques

- Single-pass table-driven with stacks
  - Bauer and Samelson for Alcor
  - Dijkstra 1960, Algol for X-1
  - Randell 1962, Whetstone Algol

- Single-pass recursive descent
  - Lucas 1961, using explicit stack
  - Hoare 1962, one procedure per language construct

- Multi-pass ad hoc
  - Fortran I, 6 passes

- Multi-pass table-driven with stacks
  - Naur 1962 GIER Algol, 9 passes
  - Hawkins 1962 Kidsgrove Algol

- General syntax-directed table-driven
  - Irons 1961 Algol for CDC 1604
History: Run-time organization

- Early papers focus on translation
  - Runtime data management was trivial, eg. Fortran I
- Algol: runtime storage allocation is essential
- Dijkstra: Algol for X-1 (1960)
  - Runtime stack of procedure activation records
  - Display, to access variables in enclosing scopes
- Also focus of Naur's Gier Algol papers, and Ekman's thesis on SMIL Algol

- Design a runtime state structure (invariant)
- Compiler should generate code that
  - Can rely on the runtime state invariant
  - Must preserve the runtime state invariant
Fortran I, 1957

- John Backus and others at IBM USA
- Infix arithmetics, mathematical formulas
- Structurally very primitive language
  - Simple function definitions, no recursion
  - No procedures
  - No scopes, no block structure
- Extremely ambitious compiler optimizations
  - common subexpression elimination
  - constant folding
  - fast index computations: reduction in strength
  - clever allocation of index registers
  - Monte Carlo simulation of execution frequencies (!)
- Large and slow compiler, 8 cards/minute
Algol 60, chiefly Europe

• Dijkstra NL, Bauer DE, Naur DK, Hoare UK, Randell UK, ... but also US, 1958-1962

• Beatiful "modern" programming language
  – Procedures, functions and recursion
  – Procedures as parameters to procedures
  – Block structure, nested scopes

• Compilers generated relatively slow code
  – Few optimizations
Early Nordic hardware and autocodes

- **BESK**, Sweden 1953, government research
  - By Stemme and others, based on IAS machine design
  - 4 bit binary-only code (Dahlquist, Dahlstrand)
  - FA-4 and FA-5 autocode, Hellström 1956, loader
  - Alfakod, symbolic no infixes, Riesel et al 1958

- **Ferranti Mercury**, Norway 1957, defense research
  - Commercial, first machine delivered, 1m NOK
  - MAC, Mercury Autocode by O-J Dahl, arrays, indexing, infix
    - Not used elsewhere
    - Independent of Brooker, Manchester Autocode, 1956-1958

- **DASK**, Denmark 1958, government research
  - By Scharøe and others, based on BESK + index registers
  - Naur EDSAC-inspired symbolic loader, 5 bit, 1957
  - No need for a more complex autocode
  - Instead an Algol 60 compiler (though without recursion)
Example problem

• Compute the polynomial
  \[ f(x) = a_0 x^8 + a_1 x^7 + \ldots + a_7 x + a_8 \]
  using Horner's rule

• In Java or C or C++ or C# anno 2014:

```java
res := 0.0;
for (i = 0; i <= 8; i++)
    res = a[i] + x * res;
```
BESK FA-5 and Alfakod, Stockholm

- Input on 4-bit paper tape (hexadecimal) only
- Hellström & Dahlquist, FA-5 1956
- Riesel et al, Alfakod 1957

BESK hex. code 1953

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>18050</td>
</tr>
<tr>
<td>000001</td>
<td>20407</td>
</tr>
<tr>
<td>000002</td>
<td>00648</td>
</tr>
<tr>
<td>000003</td>
<td>00863</td>
</tr>
<tr>
<td>000004</td>
<td>FFF28</td>
</tr>
<tr>
<td>000005</td>
<td>20446</td>
</tr>
<tr>
<td>000006</td>
<td>1820B</td>
</tr>
<tr>
<td>000007</td>
<td>2034E</td>
</tr>
<tr>
<td>000008</td>
<td>00001</td>
</tr>
<tr>
<td>000009</td>
<td>18431</td>
</tr>
<tr>
<td>00000A</td>
<td>3000C</td>
</tr>
</tbody>
</table>

FA-5 1956

- 18050
- 20407
- 00648
- A203
- A204
- 00863
- FFF28
- 20446
- 1820B
- 2034E
- 00001
- 18431
- 3000C

Alfakod 1958

- NOL AR
- FIX 0, I
- 1 MUL X
- ADD Y/I
- ADX 1, I
- VMX 1, I, 8

Dahlstrand: 2009: Minnen
Dahlquist: 1956: KodningFor
Hellstroem: 1958: KodningMed
Riesel: 1958: AlfakodningFor
Ferranti Mercury, NDRE Oslo

- Dahl, MAC=Mercury Autocode 1957
- Note
  - Infix arithmetic, logical expressions
  - Symbolic labels such as \( (1 \)
  - Real and complex numbers, arrays (1D, 2D, 3D)
  - Array index expressions \( \text{Un1} \) with optimization

\[
\begin{align*}
0 & \rightarrow A \\
0 & \rightarrow n1 \\
\text{Un1 + X A} & \rightarrow A \\
n1 + 1 & \rightarrow n1 \\
n1 < 9 & \ ? \ JUMP1
\end{align*}
\]
DASK loader, Copenhagen

• Naur, EDSAC-inspired external code, 1957
• Naur was unimpressed with the BESK code:

> har aldrig forstået fordelaten ved dette system. På Besk er det en pinlig nødvendighed, på grund af det rudimentære indlæsesystem

DASK code 1958

Naur:1957:DaskOrdrekode

DASK Algol 1961

Andersen:1958:LaerebogI

Naur:1964:RevisedReport

res := 0;
for i := 0 until 8 do
  res := a[i] + x * res;

Index register, not self-modifying
Early Nordic (Algol) compilers

- Naur, Jensen, Mondrup, in Copenhagen
  - DASK Algol 1961, no recursion
  - GIER Algol 1962
- Dahlstrand and Laryd, in Gothenburg (Facit)
  - FACIT Algol 1961, no recursion, based on Naur ...
  - SAAB Algol 1963
- Ekman, in Lund
  - SMIL Algol 1962, no recursion
- Dahl and Nygaard, in Oslo
  - Simula 1965, based on Univac 1107 Algol from US
  - First object-oriented language
  - Extremely influential: Smalltalk, C++, Java, C#…
The nuclear origins of OO

- Garwick, Nygaard, Dahl at NDRE, the Norwegian Deference Research Establishment
- Norway 6th country to have a nuclear reactor
  - in November 1951
  - six years before Risø in Denmark
- Garwick and Nygaard computed parts of the reactor design 1947-1951
  - w Monte Carlo methods to simulate neutron flow
  - chiefly hand calculators
- Ole-Johan Dahl hired 1952
  - developed "programs" for modified Bull mech. calc.
  - from 1957 developed MAC autocode for Ferranti

References:
- Randers:1946:RapportTil
- Forlan:1987:PaaLeiting
- Forlan:1997:NorwaysNuclear
- Holmevik:1994:CompilingSimula
- Holmevik:2005:InsideInnovation
- Garwick:1947:Kritisk
- Garwick:1951:BeregningAv
- Nygaard:1952:OnThe
## Norway: Nuclear and computing 1946-1962

Norway’s nuclear and computing pioneers

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Astrophysics</th>
<th>GRan</th>
<th>JGar</th>
<th>JGar</th>
<th>JGar</th>
<th>JGar</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDRE/IFA</td>
<td></td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
</tr>
<tr>
<td>NCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USA</td>
<td></td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
</tr>
<tr>
<td>UK</td>
<td></td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
<td>GRan</td>
</tr>
<tr>
<td>France</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Garwick**: 1947: Kritisk
- **Holmevik**: 2005: InsideInnovation
- **Forlan**: 1997: NorwaysNuclear
- **Forlan**: 1987: PaaLeiting
- **Randers**: 1946: RapportTil
Recommended reading

• Secondary sources
  – Knuth:1977:TheEarly
  – Bauer:1974:HistoricalRemarks
  – Ershov:1976:Addendum
  – Randell:1964:Algol60Implementation sec 1.2, 1.3

• Primary sources
  – Backus:1957:TheFortran
  – Samelson:1960:SequentialFormula
  – Dijkstra:1960:RecursiveProgramming
  – Naur:1963:TheDesign1
  – Naur:1965:CheckingOf
  – Randell:1964:Algol60Implementation
Thanks to

• Christian Gram, Dansk Datahistorisk Foreningen
• Robert Glück, DIKU, Copenhagen University
• Birger Møller-Petersen, Oslo University
• Knut Hegna, Oslo University Library
• Bjørg Asphaug, NDRE Library, Oslo
• Ingemar Dahlstrand, Lund University
• Torgil Ekman, Lund University
• Mikhail Bulyonkov, Russian Ac. Sci. Novosibirsk
• Christine di Bella, IAS Archives, Princeton
• George Dyson, Bellingham WA, USA
• Peter du Rietz, Tekniska Museet, Stockholm
• Hans Riesel, Uppsala University
• Dag Belsnes, Oslo
• Peter Naur, Copenhagen University
• Norman Sanders, UK