Early History of FORTRAN: The Making of a Wonder

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

Nov 2013

Outline

- Computing Before FORTRAN
- The Creation of FORTRAN
- FORTRAN I: The Language
- FORTRAN I: The Compiler
- Conclusions

Part 1

Computing Before FORTRAN

Pioneers of Programming Languages (Knuth-Pardo, 1976)

Zuse (Plankalkul, 1945)
Curry (Composition, 1948)
Rutishauser (1951)
Bohm (1951)
Glennie (AUTOCODE, 1952)
Laning/Zierler (1953)
Hopper et al. (A-2, 1953)
Ershov (P.P., 1955)
Blum (ADES, 1956)
Perlis et al. (IT, 1956)

Mauchly et al. (Short Code, 1950)
Burks (Intermediate PL, 1950)
Goldstine/von Neumann (Flow Diagrams, 1946)
Brooker (Mark I Autocode, 1954)
Kamynin/Liubimskii (P.P., 19654)
Grems/Porter (Bacaic, 1955)
Elsworth et al. (Kompiler 2, 1955)
Katz et al. (MATH-MATIC, 1956-1958)
Hopper et al. (FLOW-MATIC, 1956-1958)
Bauer/Samelson (1956-1958)

Pioneers of Programming Languages (Knuth-Pardo, 1976)

Zuse (Plankalkul, 1945)
Curry (Composition, 1948)
Rutishauser (1951)
Bohm (1951)
Glennie (AUTOCODE, 1952)
Laning/Zierler (1953)
Hopper et al. (A-2, 1953)
Ershov (P.P., 1955)
Blum (ADES, 1956)
Perlis et al. (IT, 1956)

Mauchly et al. (Short Code, 1950)
Burks (Intermediate PL, 1950)
Goldstine/von Neumann (Flow Diagrams, 1946)
Brooker (Mark I Autocode, 1954)
Kamynin/Liubimskii (P.P., 19654)
Grems/Porter (Bacaic, 1955)
Elsworth et al. (Kompiler 2, 1955)
Katz et al. (MATH-MATIC, 1956-1958)
Hopper et al. (FLOW-MATIC, 1956-1958)
Bauer/Samelson (1956-1958)

Pioneers of Programming Languages (Knuth-Pardo, 1976)

Zuse (Plankalkul, 1945)
Curry (Composition, 1948)
Rutishauser (1951)
Bohm (1951)
Glennie (AUTOCODE, 1952)
Laning/Zierler (1953)
Hopper et al. (A-2, 1953)
Ershov (P.P., 1955)
Blum (ADES, 1956)
Perlis et al. (IT, 1956)

Mauchly et al. (Short Code, 1950)
Burks (Intermediate PL, 1950)
Goldstine/von Neumann (Flow Diagrams, 1946)
Brooker (Mark I Autocode, 1954)
Kamynin/Liubimskii (P.P., 19654)
Grems/Porter (Bacaic, 1955)
Elsworth et al. (Kompiler 2, 1955)
Katz et al. (MATH-MATIC, 1956-1958)
Hopper et al. (FLOW-MATIC, 1956-1958)
Bauer/Samelson (1956-1958)

- Many efforts, and yet a breakthrough had to wait for Backus and his team
- We need to go back into the history to understand why it was so

Computing: Hand to Hand Combat with Machine (1)

- Computing was a black art
- Things available:

  The problem, the machine, the manual, and individual creativity
- *"Computers were pretty crazy things. They had very primitive instructions and extremely bizarre input-output facilities."*
- Example: Selective Sequence Electronic Calculator (SSEC), 1948 - 1952

  Store of 150 words, Vacuum tubes and electro-mechanical relays

Computing: Hand to Hand Combat with Machine (2)

- The story of paper tape
  - Punched paper tape glued to form a paper loop
  - Problem would appear and then disappear
  - Pattern repeated many times
  - Mobius strip

(Image source: Wikipedia)

- Debugging by the ear. When IBM 701 Defence Calculator arrived
  "How are we going to debug this enormous silent monster"

Beliefs of the Times

- Popular Mechanics Prediction in 1949

  Computers in the future may weigh no more than 1.5 tons

  (ENIAC, completed in 1947 weighed almost 30 tons)

- Editor of Prentice Hall business books, 1957

  I have travelled the length and breadth of this country and talked with the best people, and I can assure you that data processing is a fad that wont last out the year

Octal Humour

- *"Why cant programmers tell the difference between Christmas and New Years Eve? Because 25 in decimal is 31 in octal."*

- *"We programmed it in octal. Thinking I was still a mathematician, I taught myself to add, subtract, and multiply, and even divide in octal. I was really good, until the end of the month, and then my check book didn't balance! It stayed out of balance for three months until I got hold of my brother who was a banker. After several evenings of work he informed me that at intervals I had subtracted in octal. And I faced the major problem of living in two different worlds."*

  "That may have been one of the things that sent me to get rid of octal as far as possible."

  – Grace Hopper

The Priesthood of Computing

- *"Programming in the America of the 1950s had a vital frontier enthusiasm virtually untainted by either the scholarship or the stuffiness of academia."*
- *"Programmer inventors of the early 1950s were too impatient to hoard an idea until it could be fully developed and a paper written. They wanted to convince others. Action, progress, and outdoing one's rivals were more important than mere authorship of a paper."*
- *"An idea was the property of anyone who could use it and the scholarly practice of noting references to sources and related work was almost universally unknown or unpractised."*

Obstacles in Creation of a High Level Language

- Priesthood wanted to preserve the order

  "Priesthood wanted and got simple mechanical aids for the clerical drudgery which burdened them, but they regarded with hostility and derision more ambitious plans to make programming accessible to a larger population. To them, it was obviously a foolish and arrogant dream to imagine that any mechanical process could possibly perform the mysterious feats of invention required to write an efficient program."

Obstacles in Creation of a High Level Language

- Priesthood wanted to preserve the order

  "Priesthood wanted and got simple mechanical aids for the clerical drudgery which burdened them, but they regarded with hostility and derision more ambitious plans to make programming accessible to a larger population. To them, it was obviously a foolish and arrogant dream to imagine that any mechanical process could possibly perform the mysterious feats of invention required to write an efficient program."

- There also were purveyors of snake oil

  "The energetic public relations efforts of some visionaries spread the word that their "automatic programming" systems had almost human abilities to understand the language and needs of the user; whereas closer inspection of these same systems would often reveal a complex, exception-ridden performer of clerical tasks which was both difficult to use and inefficient."

The A2 Compiler

- Programmers had a library of subroutine
- They needed to copy the subroutine on the coding sheets by hand and change addresses manually
- Grace Hopper added a "call" operation whereby
  - the machine would copy the code
  - and update the addresses
- Inspiration for implementing a forward jump: A game of basketball!
- The name "compiler" was used because it put together a set of subroutines

The "Real" High Level Languages

- Conrad Zuse's Plankalkul developed in a small village in Germany (1945)
  - "Program Calculus"
  - Only design, no implementation
    (Computers were destroyed in world war II)
- Laning and Zierler's language for the WHIRLWIND at MIT (1953)
  - Fully algebraic in terms of supporting expressions
  - Very inefficient

Challenges for Creation of High Level Languages

- The tyranny of OR

  Expressiveness OR Efficiency
- Expressiveness:

  Higher level abstraction, features not supported by hardware
- Most time was spent in floating point subroutines
  - Not much attention was paid to address calculation, good use of registers

Challenges for Creation of High Level Languages

- The tyranny of OR

  Expressiveness OR Efficiency
- Expressiveness:

  Higher level abstraction, features not supported by hardware
- Most time was spent in floating point subroutines
  - Not much attention was paid to address calculation, good use of registers
- IBM 704 directly supported fast floating point operations
  - One need of expressiveness vanished revealing inefficiencies
    Clumsy treatment of loops, indexing, references to registers
  - Led to rejection of "automatic programming"

Part 2

The Creation of FORTRAN

The Genius of John Backus

He made the following important observations

- The main reason of inefficiency was a clumsy treatment of loops and array address computations

  If that could be handled, things may be far different
- The possibility made a lot of economic sense
- Language implementation was far more critical than language design

The Genius of John Backus

He made the following important observations

- The main reason of inefficiency was a clumsy treatment of loops and array address computations

  If that could be handled, things may be far different

- The possibility made a lot of economic sense

- Language implementation was far more critical than language design

  The "TRAN" in "FORTRAN" conveys the spirit

The Genesis of FORTRAN

- Motivation:

  Programming and debugging costs already exceeded the cost of running a program, and as computers became faster and cheaper this imbalance would become more and more intolerable

- Goals: Can a machine translate
  - a sufficiently rich mathematical language into
  - a sufficiently economical program at
  - a sufficiently low cost

  to make the whole affair feasible?

  The generated programs needed to be comparable to hand coded programs in efficiency

The Design Philosophy

- About Language Design
  - *"We simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs."*
  - *"We had notions of assignment statements, subscripted variables, and the DO statement as the main features. Whatever else was needed emerged as we tried to build a way of programming on these basic ideas."*

The Design Philosophy

- About Language Design
  - *"We simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs."*
  - *"We had notions of assignment statements, subscripted variables, and the DO statement as the main features. Whatever else was needed emerged as we tried to build a way of programming on these basic ideas."*
- About Compiler Design
  - Study the inner loops to find the most efficient method of execution
  - Find how the efficient code can be generated for sample statements
  - Generalize the observations by removing specificities and exceptions

The Design Philosophy

- About Language Design
  - *"We simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs."*
  - *"We had notions of assignment statements, subscripted variables, and the DO statement as the main features. Whatever else was needed emerged as we tried to build a way of programming on these basic ideas."*
- About Compiler Design
  - Study the inner loops to find the most efficient method of execution
  - Find how the efficient code can be generated for sample statements
  - Generalize the observations by removing specificities and exceptions

Effectively, they raised the level of computing from

number processing to processing text that processed numbers

The FORTRAN Project

- Approved in Jan 1954, system delivered in April 1957
- Supportive management
- Young, energetic, enthusiastic, and inexperienced team
  - Great team spirit and synergy

The FORTRAN Project

- Approved in Jan 1954, system delivered in April 1957
- Supportive management
- Young, energetic, enthusiastic, and inexperienced team
  - Great team spirit and synergy

    "The best part was the uncertainty and excitement of waiting to see what kinds of object code all that work was finally going to produce."

The FORTRAN Project

- Approved in Jan 1954, system delivered in April 1957
- Supportive management
- Young, energetic, enthusiastic, and inexperienced team
  - Great team spirit and synergy

    "The best part was the uncertainty and excitement of waiting to see what kinds of object code all that work was finally going to produce."

    "It was great sport in those days to scan the object program and either marvel at the translator or question its sanity!"

The FORTRAN Project

- Approved in Jan 1954, system delivered in April 1957
- Supportive management
- Young, energetic, enthusiastic, and inexperienced team
  - Great team spirit and synergy

    "The best part was the uncertainty and excitement of waiting to see what kinds of object code all that work was finally going to produce."

    "It was great sport in those days to scan the object program and either marvel at the translator or question its sanity!"

  - Helped in ignoring the doubters and overcome discouragement and despair

FORTRAN Claims

- *"The amount of knowledge necessary to utilize the 704 effectively by means of FORTRAN is far less than the knowledge required to make effective use of the 704 by direct coding.*

  It will be possible to make the full capabilities of the 704 available to a much wider range of people than would otherwise be possible without expensive and time-consuming training programs."

FORTRAN Claims

- *"The amount of knowledge necessary to utilize the 704 effectively by means of FORTRAN is far less than the knowledge required to make effective use of the 704 by direct coding.*

  It will be possible to make the full capabilities of the 704 available to a much wider range of people than would otherwise be possible without expensive and time-consuming training programs."
- *"FORTRAN may apply complex, lengthy techniques in coding a problem which the human coder would have neither the time nor inclination to derive or apply."*

FORTRAN Claims

- *"The amount of knowledge necessary to utilize the 704 effectively by means of FORTRAN is far less than the knowledge required to make effective use of the 704 by direct coding.*

  It will be possible to make the full capabilities of the 704 available to a much wider range of people than would otherwise be possible without expensive and time-consuming training programs."
- *"FORTRAN may apply complex, lengthy techniques in coding a problem which the human coder would have neither the time nor inclination to derive or apply."*
- *"FORTRAN will virtually eliminate coding and debugging."* ☺

Part 3

FORTRAN I: The Language

The Very First Question in FORTRAN FAQ

In the IBM Customer Engineering Manual of Instructions

- Q. Why is Fortran used and what are its advantages over the SHARE assembly program ?
- A. Fortran allows a programmer to write in relatively familiar and simple language the steps of a procedure to be carried out by the 704. The programmer need not know 704 language, and is relieved of clerical work; human error is minimized. The programmer writes in symbolic machine language in SHARE. Fortran translates, compiles, and assembles, whereas a SHARE assembly program essentially just assembles, although subroutines can be compiled from the library tape of SHARE.

The Language FORTRAN

- Scalar and array variables
- Integer and real (floating point) values
- Expressions
- Assignment statements
- DO loops
- Functions
- Other statements: READ, PRINT, FORMAT, IF and GOTO
- Comments

FORTRAN Examples (1)

Formula	$root = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$
FORTRAN Statement	`ROOT = (-B + SQRTF(B**2 - 4*A*C))/(2.0*A)`
Defining Function	`ROOTF(A,B,C) = (-B + SQRTF(B**2 - 4*A*C))/(2.0*A)`

FORTRAN Examples (2)

Problem:
Set Q_{max} equal to the largest quantity $\frac{P(a_i+b_i)}{P(a_i-b_i)}$ for some i between 1 and 1000 where $P(x) = c_0 + c_1x + c_2x^2 + c_3x^3$

FORTRAN Program

```
1   POLYF(X) = C0+X*(C1+X*(C2+X*C3))
2   DIMENSION A(1000), B(1000)
3   QMAX = -1.0E20
4   Do 5 I = 1, 1000
5   QMAX = MAXF(QMAX, POLYF(A(I) + B(I))/POLYF(A(I) - B(I)))
6   STOP
```

Limitations of FORTRAN I Language

- No reserved words
- Simplistic functions
- No subprograms, no recursion
- No spaces
- DO loops with limited nesting depth of 3
- Implicit types based on the first letter
- No declarations required

Minor Errors Could be Rather Expensive

- The first American Venus probe was lost because of a computer problem
- A programmer replaced a comma by a dot

Should have been	Was
`DO 3 I = 1, 3`	`DO 3 I = 1.  3`

- What was essentially a DO loop header got treated as an assignment statement `DO3I = 1.3` by the compiler

Fun with FORTRAN

- A provision to override the default types was added later

- No reserved words

Fun with FORTRAN

- A provision to override the default types was added later

  "GOD is real unless declared integer".

- No reserved words

  IF (IF .LT. THEN) THEN ELSE = THEN ELSE THEN = ELSE

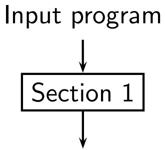
Part 4

FORTRAN I: The Compiler

Contributions of FORTRAN I Compiler

- Phase-wise division of work
- Optimizations:
  - Common subexpressions elimination,
  - Array address optimization in loops
    (a form of strength reduction and induction variable elimination)
  - Register allocation using hierarchical regions
    (optimal under number of loads for straight line code)
- Basic blocks and execution frequency analysis
- Distinction between pseudo registers and hard registers

Phases of FORTRAN I Compiler

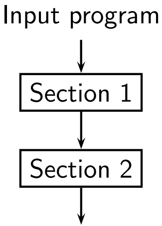


- Input may be on tape or cards
- Transferred to tape 2
- Statements are classified and Internal Formula Number (IFN) is assigned
- Arithmetic statements are translated
- Output is recorded on COMPAIL file on tape 2

  (Complete Arithmetic, Input-Output, Logical)
- Other statements are stored in buffer areas

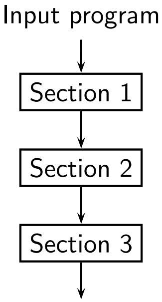
  (if it is full, the information is transferred to tape 4)

Phases of FORTRAN I Compiler



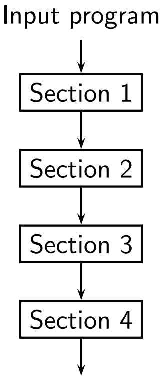
- DO loops are translated
- Arithmetic statements involving subscripts and induction variables are translated
- Unlimited index registers are assumed (in place of actual 3 index registers)
- Output is recorded on COMPDO file on tape 2

Phases of FORTRAN I Compiler



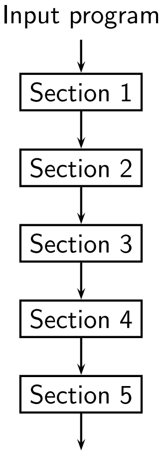
- COMPAIL and COMPDO files are merged into a single file
- Rest of the statements are translated
- Translation is complete except that actual index registers are not used

Phases of FORTRAN I Compiler



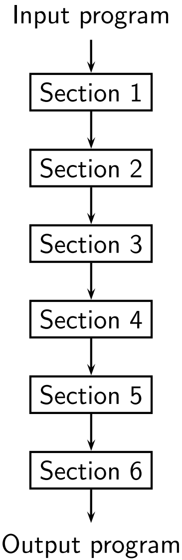
- Basic blocks are created and flow analysis is performed
- Execution frequencies are computed using simulated execution
- The program may be executed several hundred times
- Outcome of conditional control transfers is determined by
  - a random number generator suitably weighted according to
  - the branch frequency specification in the program

Phases of FORTRAN I Compiler



- Pseudo registers are replaced by hard index register
- Results of flow analysis of section 5 are used
- Hierarchical regions are formed and inner most regions are assigned the registers first
- "Distance-to-next-use" policy is used to evict registers if required
- Now the translation to assembly is complete

Phases of FORTRAN I Compiler

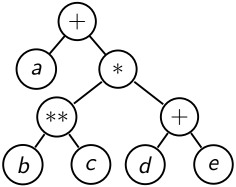


- The program is assembled to produce the executable
- It may be created on the tape or on cards
- A listing of the program can also be generated

  Source statements and corresponding executable statements

Expressions in the Programs

- Other "algebraic" compilers needed parenthesis for expressions
- No concept for parsing using grammars

Expression	Expression Tree	Required Syntax
$a + b * *c * (d + e)$		$(a) + (b * *(c * (d + e)))$

FORTRAN Rules for Expressions

1. Any fixed point (floating point) constant, variable, or subscripted variable is an expression of the same mode. Thus 3 and I are fixed point expressions, and $ALPHA$ and $A(I, J, K)$ are floating point expressions.
2. If $SOMEF$ is some function of n variables, and if $E, F, \ldots, H$ are a set of n expressions of the correct modes for $SOMEF$, then $SOMEF(E, F, \ldots, H)$ is an expression of the same mode as $SOMEF$.
3. If E is an expression, and if its first character is not "+" or "−", then $+E$ and $-E$ are expressions of the same mode as E. Thus $-A$ is an expression, but $- - A$ is not.
4. If E is an expression, then (E) is an expression of the same mode as E. Thus $(A), ((A)), (((A)))$, etc. are expressions.
5. If E and F are expressions of the same mode, and if the first character of F is not $+$ or $-$, then $E + F$, $E - F$, $E * F$, E/F are expressions of the same mode.

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "$+$", "$*$", and "$**$"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "$+$" by "$))) + ((($"
  - Replace every "$*$" by "$)) * (($"
  - Replace every "$**$" by "$) * *($"

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "$+$", "$*$", and "$**$"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "$+$" by "$))) + ((($"
  - Replace every "$*$" by "$)) * (($"
  - Replace every "$**$" by "$) ** ($"
- Our expression becomes fully parenthesized by application of this rule.

$$A \quad + \quad B ** C \quad * \quad (D \quad + \quad E)$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "+", "*", and "**"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "+" by ")))+((("
  - Replace every "*" by "))*(("
  - Replace every "**" by ")**("
- Our expression becomes fully parenthesized by application of this rule.

$$(((A \quad + \quad B ** C \quad * \quad (D \quad + \quad E)$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "+", "*", and "**"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "+" by ")))+((("
  - Replace every "*" by "))*(("
  - Replace every "**" by ")**("
- Our expression becomes fully parenthesized by application of this rule.

$$(((A \quad + \quad B ** C \quad * \quad ((((D \quad + \quad E)$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "$+$", "$*$", and "$**$"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "$+$" by "$))) + ((($"
  - Replace every "$*$" by "$)) * (($"
  - Replace every "$**$" by "$) ** ($"
- Our expression becomes fully parenthesized by application of this rule.

$$(((A \quad + \quad B ** C \quad * \quad ((((D \quad + \quad E) \quad)))$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "+", "∗", and "∗∗"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "+" by ")))+((("
  - Replace every "∗" by "))∗(("
  - Replace every "∗∗" by ")∗∗("
- Our expression becomes fully parenthesized by application of this rule.

$$(((A \quad + \quad B ** C \quad * \quad ((((D \quad + \quad E)))))))$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "+", "*", and "**"

  - Add "(((" in the beginning of the expression (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression (and hence after every ")" in the expression)
  - Replace every "+" by ")))+((("
  - Replace every "*" by "))*(("
  - Replace every "**" by ")**("
- Our expression becomes fully parenthesized by application of this rule.

$$(((A))) + (((B \;\; ** \;\; C \quad * \quad ((((D))) + (((E)))))))$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "$+$", "$*$", and "$**$"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "$+$" by "$))) + ((($"
  - Replace every "$*$" by "$)) * (($"
  - Replace every "$**$" by "$) ** ($"
- Our expression becomes fully parenthesized by application of this rule.

$$(((A))) + (((B \;**\; C)) * ((((((D))) + (((E)))))))$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "$+$", "$*$", and "$**$"

  - Add "(((" in the beginning of the expression
    (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression
    (and hence after every ")" in the expression)
  - Replace every "$+$" by "$))) + ((($"
  - Replace every "$*$" by "$)) * (($"
  - Replace every "$**$" by "$) ** ($"
- Our expression becomes fully parenthesized by application of this rule.

$$(((A))) + (((B) ** (C)) * ((((((D))) + (((E)))))))$$

FORTRAN Expression Handling

- Conventional precedences were used and parenthesis were not required.
- Simple rule of reconstructing parenthesized expressions:

  Assuming three levels of precedences of "$+$", "$*$", and "$**$"

  - Add "(((" in the beginning of the expression (and hence before every "(" in the expression)
  - Add ")))" at the end of the expression (and hence after every ")" in the expression)
  - Replace every "$+$" by "$))) + ((($"
  - Replace every "$*$" by "$)) * (($"
  - Replace every "$**$" by "$) * *($"
- Our expression becomes fully parenthesized by application of this rule.

$$(((A))) + (((B) * *(C)) * ((((((D))) + (((E)))))))$$

(The rules can be applied in a single left-to-right scan of the expression)

FORTRAN Compiler Anecdotes (1)

- Expression computation problem observed by Bernard A. Galler
  - For $n = 10$, the expression $n * (n - 1)/2$ computed 40 instead of 45!

FORTRAN Compiler Anecdotes (1)

- Expression computation problem observed by Bernard A. Galler
  - For $n = 10$, the expression $n * (n - 1)/2$ computed 40 instead of 45!
  - "/" had a higher precedence and $9/2$ is 4 in integer arithmetic

FORTRAN Compiler Anecdotes (1)

- Expression computation problem observed by Bernard A. Galler
  - For $n = 10$, the expression $n * (n - 1)/2$ computed 40 instead of 45!
  - "/" had a higher precedence and $9/2$ is 4 in integer arithmetic
- Response from IBM

  "It is too complicated to change the compiler so we will fix the manual"

FORTRAN Compiler Anecdotes (1)

- Expression computation problem observed by Bernard A. Galler
  - For $n = 10$, the expression $n * (n - 1)/2$ computed 40 instead of 45!
  - "/" had a higher precedence and $9/2$ is 4 in integer arithmetic
- Response from IBM

  "It is too complicated to change the compiler so we will fix the manual"
- New manual had the following statement:

  "Please be warned that mathematical equivalence is not the same as computational equivalence"

FORTRAN Compiler Anecdotes (1)

- Expression computation problem observed by Bernard A. Galler
  - For $n = 10$, the expression $n * (n - 1)/2$ computed 40 instead of 45!
  - "/" had a higher precedence and $9/2$ is 4 in integer arithmetic
- Response from IBM

  "It is too complicated to change the compiler so we will fix the manual"
- New manual had the following statement:

  "Please be warned that mathematical equivalence is not the same as computational equivalence"
- How about the same precedence for "/" and "$*$" and left associativity?

FORTRAN Compiler Anecdotes (1)

- Expression computation problem observed by Bernard A. Galler
  - For $n = 10$, the expression $n * (n - 1)/2$ computed 40 instead of 45!
  - "/" had a higher precedence and $9/2$ is 4 in integer arithmetic
- Response from IBM

  "It is too complicated to change the compiler so we will fix the manual"
- New manual had the following statement:

  "Please be warned that mathematical equivalence is not the same as computational equivalence"
- How about the same precedence for "/" and "$*$" and left associativity?
  - $n/2 * (n - 1)$
  - $n * (n - 1) * (1/2)$

FORTRAN Compiler Anecdotes (2)

On compiler reliability

- Tables stored on the magnetic drum based memory
- Slow searches and more load on drums
- The compiler worked far better at GM than at Westinghouse
- GM people had ensured a much better servicing of magnetic drums!

FORTRAN Compiler Anecdotes (3)

On compiler efficiency

- Frank Engel at Westinghouse observed that tapes moved independently but sequentially
- Compiler could become faster if tape movement is made to overlap
- Frank asked for the source and got a reply: (source meant assembly)
  "IBM does not supply source code"
- Frank patched up the octal object code of the compiler and the throughput increased by a factor of 3!
- IBM was surprised and wanted a copy, so Frank said:
  "Westinghouse does not supply object code"

A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments

B(1,1)	B(1,2)	B(1,3)
B(2,1)	B(2,2)	B(2,3)
B(3,1)	B(3,2)	B(3,3)
B(4,1)	B(4,2)	B(4,3)

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)
A(4,1)	A(4,2)	A(4,3)

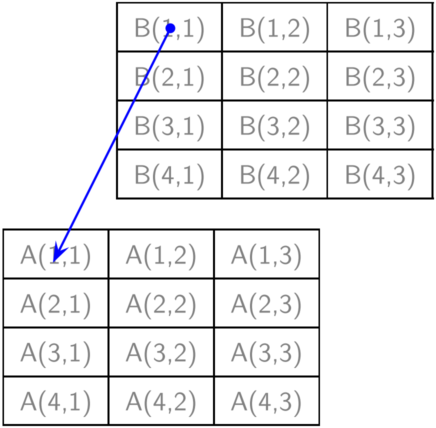
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



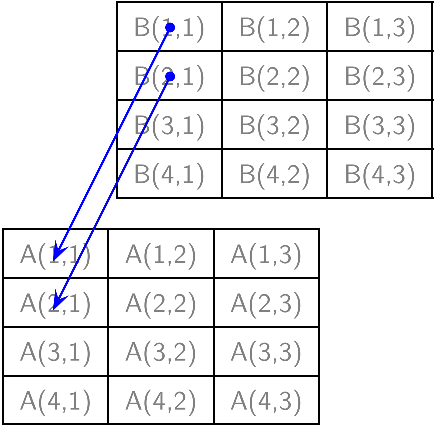
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



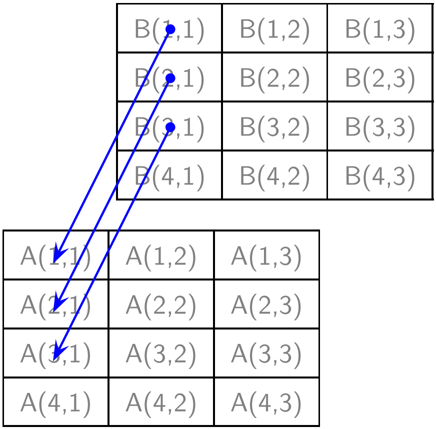
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



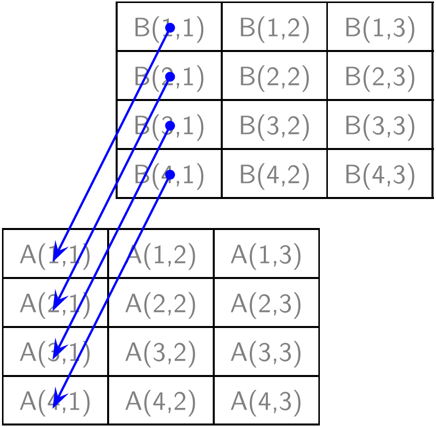
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



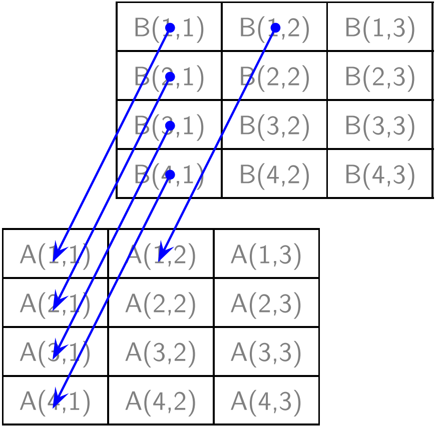
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



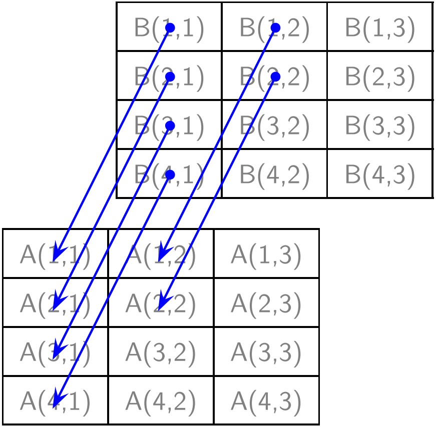
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



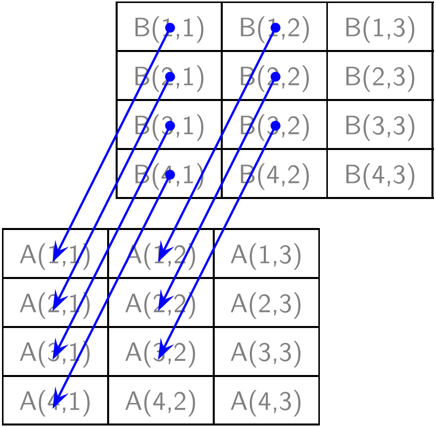
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



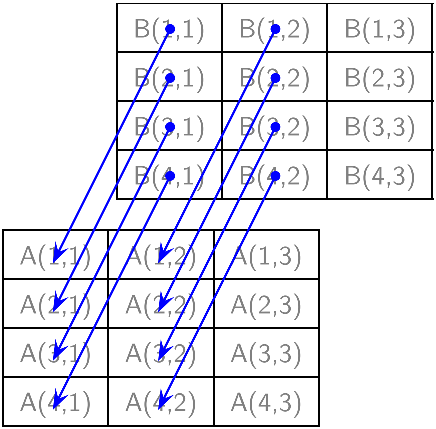
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



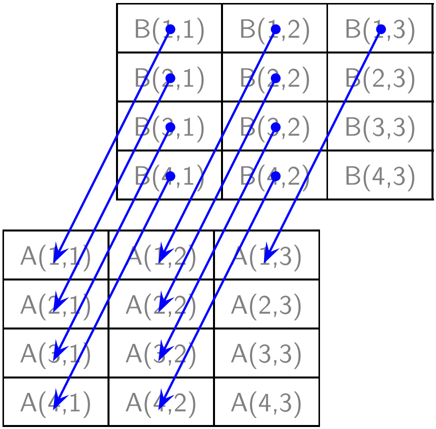
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



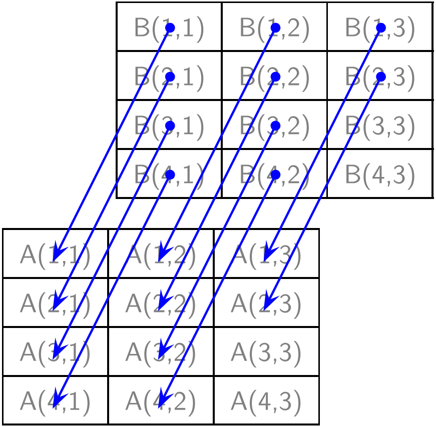
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



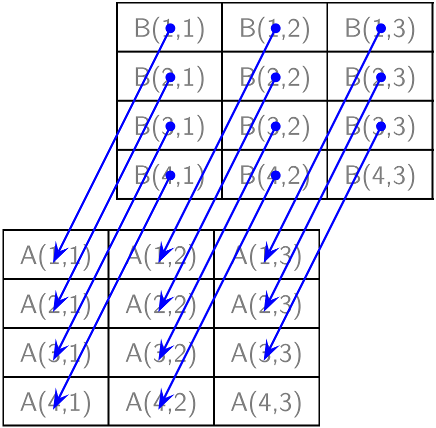
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

A simplified view for 4x3 fragments



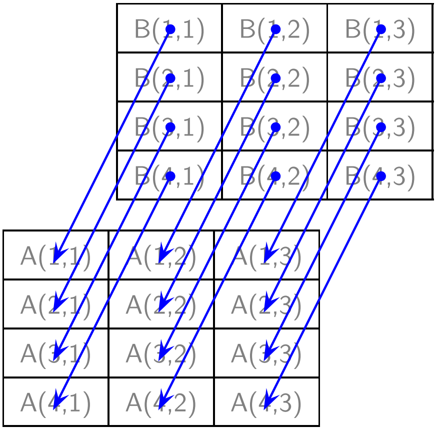
A FORTRAN Program for Array Copy

Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

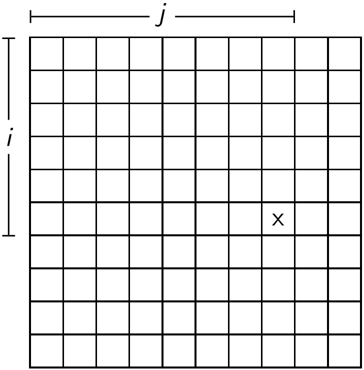
A simplified view for 4x3 fragments



Array Address Calculation

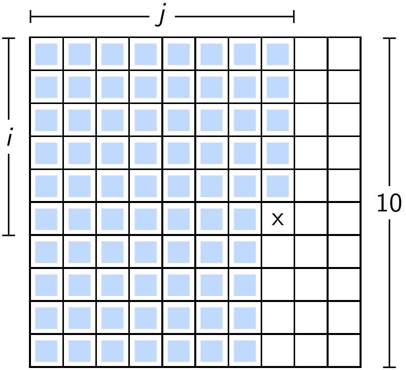
Cell (i, j)

Its address



Array Address Calculation

Cell (i, j)

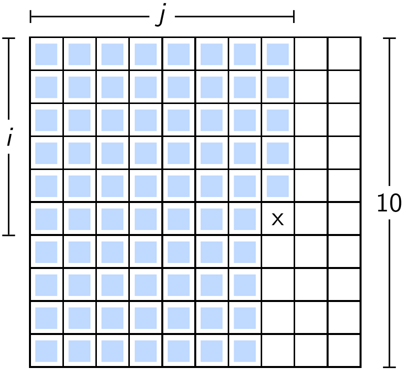


Its address

$\text{Base} + (j-1) * 10 + i - 1$

Array Address Calculation

Cell (i, j) Its address



Base $+ (j-1) * 10 + i - 1$

An additional complication: In FORTRAN, arrays are stored backwards and index registers are subtracted from the base

Output of FORTRAN I Compiler

Source Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

Object Program

	Statement	Explanation
	`LXD ONE, 1`	$Ixr1 = 1$
LOOP	`CLA B+1, 1`	$Acc = *(B + 1 - Ixr1)$
	`STO A+1, 1`	$*(A + 1 - Ixr1) = Acc$
	`TXI * +1, 1, 1`	$Ixr1 = Ixr1 + 1$, jump ahead by 1
	`TXL LOOP,1 ,100`	if ($Ixr1 \leq 100$), goto LOOP

Output of FORTRAN I Compiler

Source Program

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

- Address calculation?
- Nested loops?

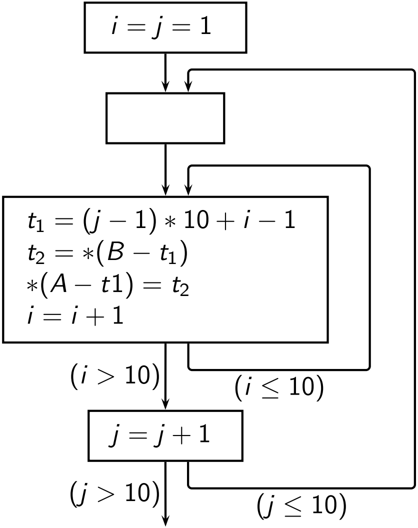
Object Program

	Statement	Explanation
	`LXD ONE, 1`	$Ixr1 = 1$
`LOOP`	`CLA B+1, 1`	$Acc = *(B + 1 - Ixr1)$
	`STO A+1, 1`	$*(A + 1 - Ixr1) = Acc$
	`TXI * +1, 1, 1`	$Ixr1 = Ixr1 + 1$, jump ahead by 1
	`TXL LOOP,1 ,100`	if ($Ixr1 \leq 100$), goto LOOP

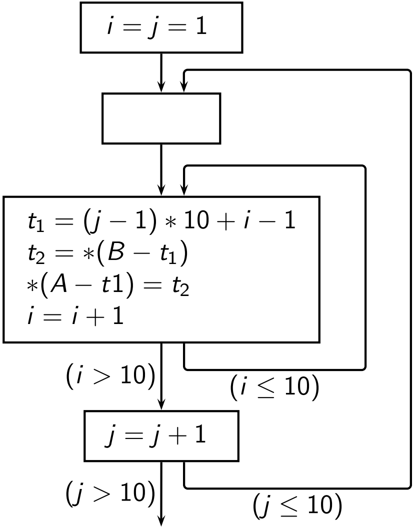
Compiling Array Copy Program: Control Flow Graph

```
      DIMENSION A (10,10)
      DIMENSION B (10,10)

      DO 1 J = 1, 10
      DO 1 I = 1, 10
1     A(I,J) = B(I,J)
```

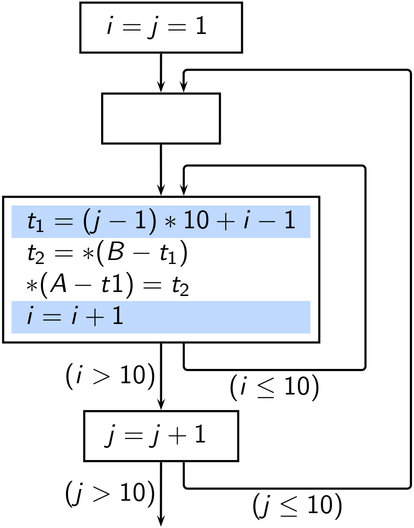


Compiling Array Copy Program: Strength Reduction (1)



Observations about the inner loop

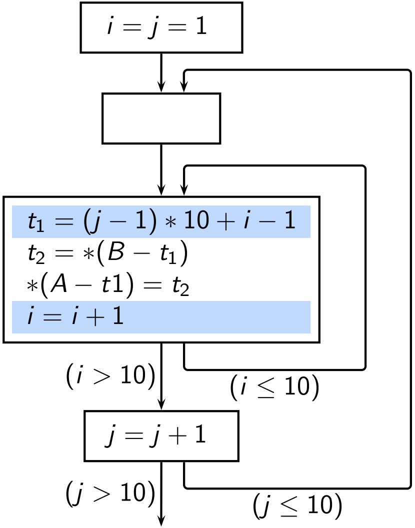
Compiling Array Copy Program: Strength Reduction (1)



Observations about the inner loop

- Whenever i increments by 1, $t1$ also increments by 1

Compiling Array Copy Program: Strength Reduction (1)



Observations about the inner loop

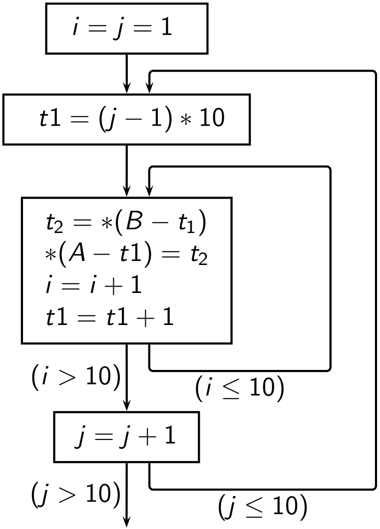
- Whenever i increments by 1, $t1$ also increments by 1
- We can initialize $t1$ outside of the inner loop

$$\begin{aligned} t1 &= (j-1)*10+i-1 \\ &= (j-1)*10 \\ &\quad \text{(because } i \text{ is 1)} \end{aligned}$$

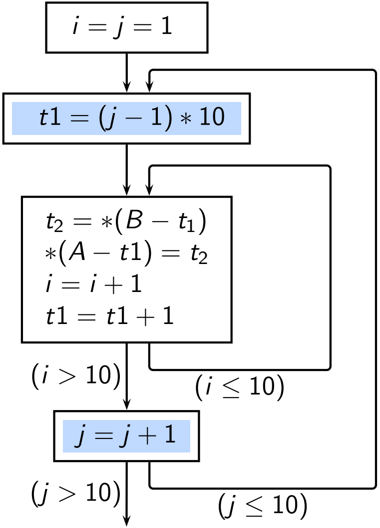
and increment it within the loop

$t1 = t1 + 1$

Compiling Array Copy Program: Strength Reduction (2)



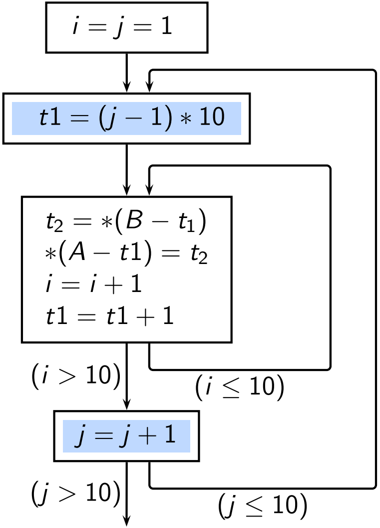
Compiling Array Copy Program: Strength Reduction (2)



Observations about the inner loop

- Whenever j increments by 1, $t1$ increments by 10

Compiling Array Copy Program: Strength Reduction (2)



Observations about the inner loop

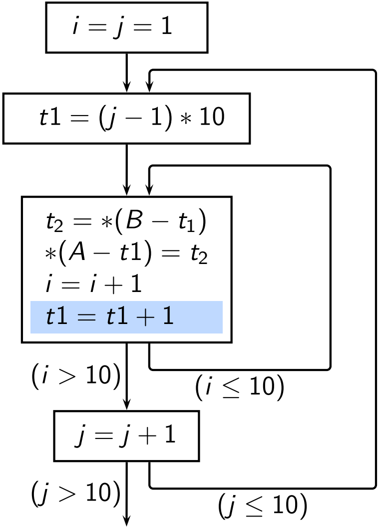
- Whenever j increments by 1, $t1$ increments by 10
- We can initialize t1 outside of the outer loop

  $$\begin{aligned} t1 &= (j-1)*10 \\ &= 0 \\ & \text{(because } j \text{ is 1)} \end{aligned}$$

  and increment it within the loop

  $t1 = t1 + 10$

Compiling Array Copy Program: Strength Reduction (2)



Observations about the inner loop

- Whenever j increments by 1, $t1$ increments by 10
- We can initialize t1 outside of the outer loop

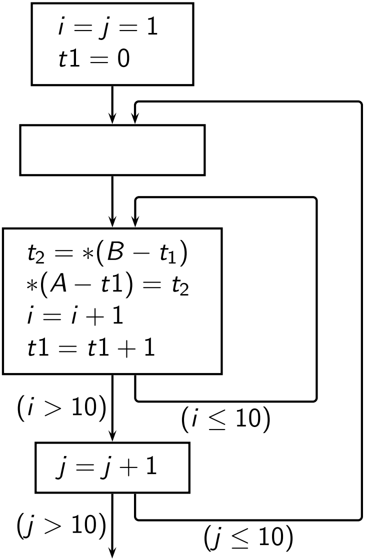
  $$t1 = (j-1) * 10 = 0 \quad \text{(because } j \text{ is 1)}$$

  and increment it within the loop

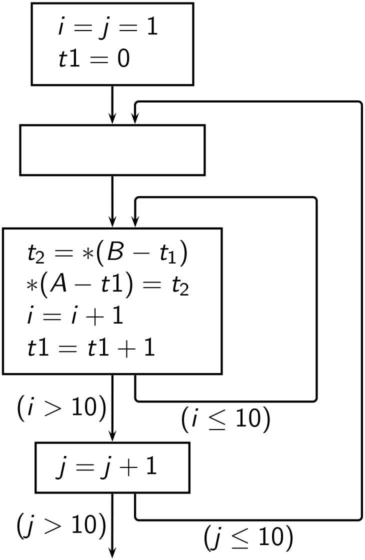
  $$t1 = t1 + 10$$

- However, the inner loop already increments $t1$ by 10.

Compiling Array Copy Program: Flattening the Loops



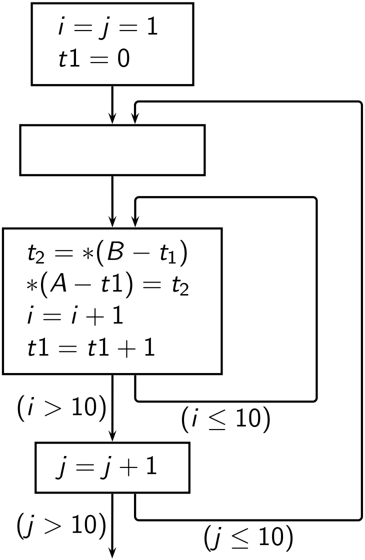
Compiling Array Copy Program: Flattening the Loops



- The only activity in the outer loop now is to control the loop iterations

  No other computation

Compiling Array Copy Program: Flattening the Loops

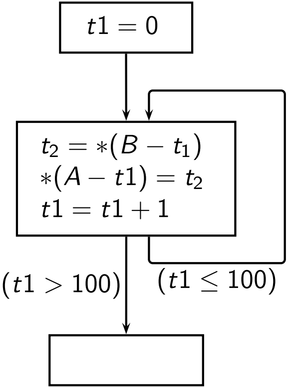


- The only activity in the outer loop now is to control the loop iterations

  No other computation

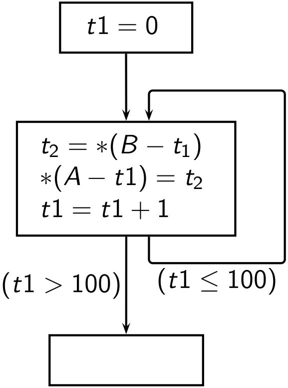
- We can combine the loops into a single loop by taking a product of the two loop bounds
- Variables i and j would not be required

Compiling Array Copy Program: The Final Program

Control flow graph (CFG)	Original Assembly
	

Compiling Array Copy Program: The Final Program

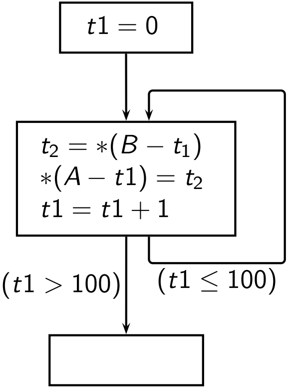
Control flow graph (CFG)	Original Assembly




```
      LXD ONE, 1
LOOP  CLA B+1, 1
      STO A+1, 1
      TXI * +1, 1, 1
      TXL LOOP,1 ,100
```

Compiling Array Copy Program: The Final Program

Control flow graph (CFG)	Original Assembly




```
        LXD ONE, 1
LOOP    CLA B+1, 1
        STO A+1, 1
        TXI * +1, 1, 1
        TXL LOOP,1 ,100
```

Minor differences

	CFG	Assembly
Base address	B	$B + 1$
Initial value of $t1$	0	1

Compiling Array Copy Program Using GCC 4.7.2 (gfortran)

```
.L5:
        leal    408(%esp), %ebx
        movl    $1, %eax
        leal    808(%esp), %ecx
        addl    %esi, %ebx
        addl    %esi, %ecx
        .p2align 4,,7
        .p2align 3
.L4:
        movl    -44(%ecx,%eax,4), %edx
        movl    %edx, -44(%ebx,%eax,4)
        addl    $1, %eax
        cmpl    $11, %eax
        jne     .L4
        addl    $40, %esi
        cmpl    $400, %esi
        jne     .L5
```

- Integer is now 4 bytes

Compiling Array Copy Program Using GCC 4.7.2 (gfortran)

```
.L5:
        leal    408(%esp), %ebx
        movl    $1, %eax
        leal    808(%esp), %ecx
        addl    %esi, %ebx
        addl    %esi, %ecx
        .p2align 4,,7
        .p2align 3
.L4:
        movl    -44(%ecx,%eax,4), %edx
        movl    %edx, -44(%ebx,%eax,4)
        addl    $1, %eax
        cmpl    $11, %eax
        jne     .L4
        addl    $40, %esi
        cmpl    $400, %esi
        jne     .L5
```

- Integer is now 4 bytes
- Efficient address calculation with strength reduction

Compiling Array Copy Program Using GCC 4.7.2 (gfortran)

```
.L5:
        leal    408(%esp), %ebx
        movl    $1, %eax
        leal    808(%esp), %ecx
        addl    %esi, %ebx
        addl    %esi, %ecx
        .p2align 4,,7
        .p2align 3
.L4:
        movl    -44(%ecx,%eax,4), %edx
        movl    %edx, -44(%ebx,%eax,4)
        addl    $1, %eax
        cmpl    $11, %eax
        jne     .L4
        addl    $40, %esi
        cmpl    $400, %esi
        jne     .L5
```

- Integer is now 4 bytes
- Efficient address calculation with strength reduction
- Nested loops not flattened

Part 5

Conclusions

So is There Nothing New in Compilers?

- Languages have changed significantly
- Processors have changed significantly
- Problem sizes have changed significantly
- Expectations have changed significantly
- Analysis techniques have changed significantly

So is There Nothing New in Compilers?

- Languages have changed significantly
  - "The worst thing that has happened to Computer Science is C because it brought pointers with it." (Frances Allen, IITK, 2007)
- Processors have changed significantly
- Problem sizes have changed significantly
- Expectations have changed significantly
- Analysis techniques have changed significantly

So is There Nothing New in Compilers?

- Languages have changed significantly
  - "The worst thing that has happened to Computer Science is C because it brought pointers with it." (Frances Allen, IITK, 2007)
- Processors have changed significantly
  - GPUs, Many core processors, Embedded processors
- Problem sizes have changed significantly
- Expectations have changed significantly
- Analysis techniques have changed significantly

So is There Nothing New in Compilers?

- Languages have changed significantly
  - "The worst thing that has happened to Computer Science is C because it brought pointers with it." (Frances Allen, IITK, 2007)
- Processors have changed significantly
  - GPUs, Many core processors, Embedded processors
- Problem sizes have changed significantly
  - Programs running in millions of lines of code
- Expectations have changed significantly
- Analysis techniques have changed significantly

So is There Nothing New in Compilers?

- Languages have changed significantly
  - "The worst thing that has happened to Computer Science is C because it brought pointers with it." (Frances Allen, IITK, 2007)
- Processors have changed significantly
  - GPUs, Many core processors, Embedded processors
- Problem sizes have changed significantly
  - Programs running in millions of lines of code
- Expectations have changed significantly
  - Interprocedural analysis and optimization, validation, reverse engineering, parallelization
- Analysis techniques have changed significantly

So is There Nothing New in Compilers?

- Languages have changed significantly
  - "The worst thing that has happened to Computer Science is C because it brought pointers with it." (Frances Allen, IITK, 2007)
- Processors have changed significantly
  - GPUs, Many core processors, Embedded processors
- Problem sizes have changed significantly
  - Programs running in millions of lines of code
- Expectations have changed significantly
  - Interprocedural analysis and optimization, validation, reverse engineering, parallelization
- Analysis techniques have changed significantly
  - Parsing, Data flow analysis, Parallism Discovery, Heap Analysis

The Wonder Element of FORTRAN

- Expressiveness Vs. Efficiency conflict
  - Efficiency of programming and reach of programming, OR
  - Efficiency of program execution and resource utilization
- FORTRAN: The triumph of the genius of AND over the tyranny of OR

The Wonder Element of FORTRAN

- Expressiveness Vs. Efficiency conflict
  - Efficiency of programming and reach of programming, OR
  - Efficiency of program execution and resource utilization
- FORTRAN: The triumph of the genius of AND over the tyranny of OR
- *The software equivalent of a transistor*

Why Things Happen the Way They Happen?

- John Backus was the *right person* at the *right time* at the *right place*

Why Things Happen the Way They Happen?

- John Backus was the *right person* at the *right time* at the *right place*
  - He had the foresight to recognize the *adjacent possible*
  - He was Bernard Shaw's proverbial "unreasonable person"

Why Things Happen the Way They Happen?

- John Backus was the *right person* at the *right time* at the *right place*
  - He had the foresight to recognize the *adjacent possible*
  - He was Bernard Shaw's proverbial "unreasonable person"
- The ideas of Charles Babbage were far beyond the adjacent possible

The Challenge Ahead

- Expressiveness Vs. Efficiency conflict due to the problem of scale

The Challenge Ahead

- Expressiveness Vs. Efficiency conflict due to the problem of scale
- Have we reached the Von Neumann bottleneck?

The Challenge Ahead

- Expressiveness Vs. Efficiency conflict due to the problem of scale
- Have we reached the Von Neumann bottleneck?
  Backus argued so over three decades ago!

The Challenge Ahead

- Expressiveness Vs. Efficiency conflict due to the problem of scale
- Have we reached the Von Neumann bottleneck?
  Backus argued so over three decades ago!
- The world awaits another John Backus to give us the next break-through!

Acknowledgements

- Mostly based on the online documents of the *Computer History Museum* (www.computerhistory.org)
  - FORTRAN examples by John Backus
  - Array copy example by Frances Allen
  - FORTRAN expression handling explanation by David Padua
- Interesting discussions with Supratim Biswas

Last But Not the Least

Thank You!

Last But Not the Least

Thank You!

Contacting me :

- `uday@cse.iitb.ac.in`
- `http://www.cse.iitb.ac.in/~uday`