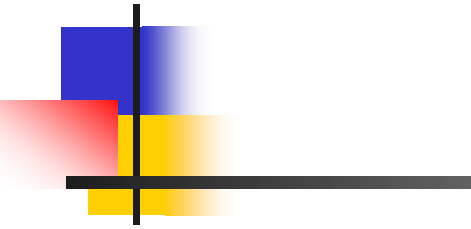
Breve Histórico & Conceitos Básicos



compiladores interpretadores montadores filtros

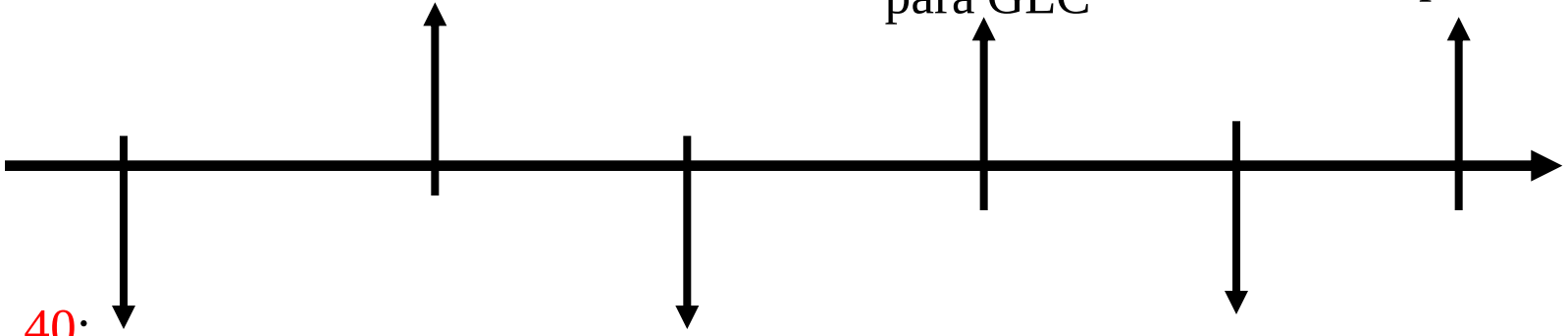
pré-processadores carregadores linkers

compilador cruzado (cross-compiler)

auto-compilável (bootstraping) auto-residente

compiler compilers (ou translator writing systems, ou compiler generators)

Linha do Tempo



40:
Computador de Programa Armazenado

54-57
Compilador Fortran

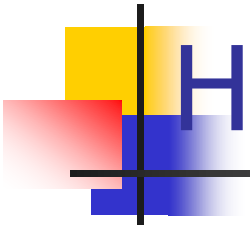
Início 60:
Hierarquia das gramáticas formais de Chomsky

60-70:
parsers eficientes para GLC

1975:

Geradores de analisadores léxicos e sintáticos – LEX e YACC

70 e 80:
automatização de outras partes do compilador



Histórico

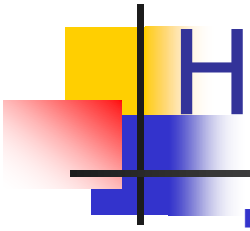
- Computador de programa armazenado (von Neumann) – final da década de 40
  - Inicialmente os programas eram escritos em código de máquina (computadores tinham pouca memória)
- Linguagem de montagem para facilitar a programação
  - Necessidade de um programa (montador) para traduzir para código de máquina
- Próximo passo: linguagem de alto nível
  - Temor: que fosse impossível ou que o código objeto fosse tão ineficiente que seria inútil!

Histórico

- Primeira Linguagem de alto nível (Fortran) e seu compilador, graças a Backus, entre 1954 e 1957
  - Muito esforço, pois grande parte dos processos de tradução ainda não eram entendidos naquele tempo,
  - Entretanto com relação a otimização de código estava a frente no seu tempo.

- Chomsky inicia estudos da estrutura da linguagem natural (inglês) no fim dos 50 – início dos 60´s
  - Complexidade das gramáticas e os algoritmos para reconhecê-las: tipo 0, 1, 2, 3.

- Determinação de algoritmos eficientes para reconhecer gramáticas livres de contexto: 1960 e 1970

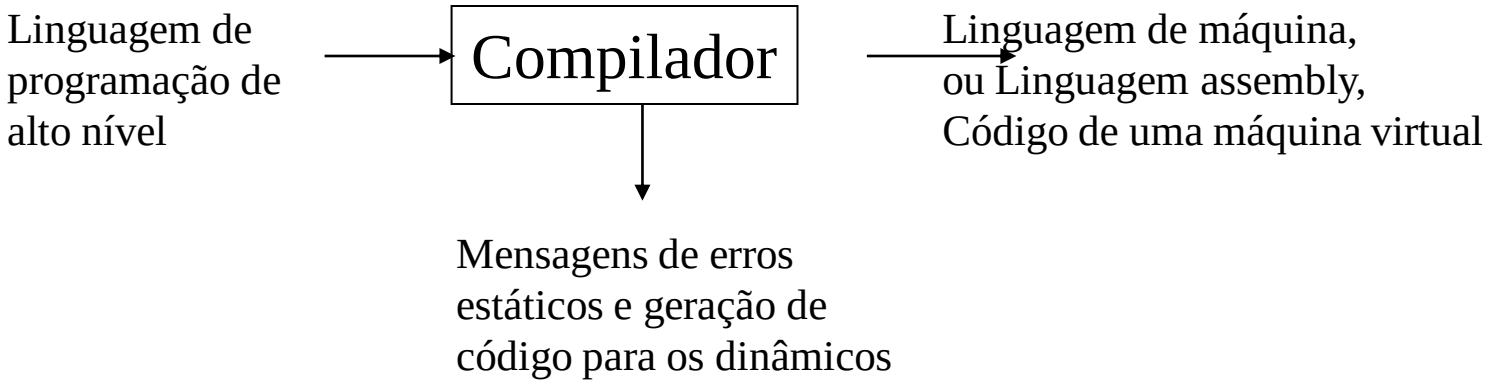


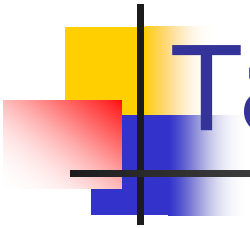
Histórico

- Vantagens da separação entre reconhecimento de tokens de uma linguagem e suas regras (sintaxe)
  - levou ao uso de métodos para expressar os tokens (uso de autômatos finitos e expressões regulares)
- Com uma maior compreensão da análise sintática surgiram métodos para automatizar partes do desenvolvimento de compiladores
  - Geradores de analisadores sintáticos (YACC – unix em 1975)
- O estudo dos autômatos finitos
  - Geradores de analisadores léxicos (Lex – unix em 1975)
- Década de 70 e 80 houve projetos para automatizar a geração de outras partes do compilador (geração de código)

Compilador

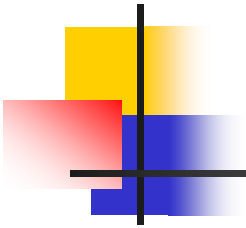
Compilador é um tradutor que transforma automaticamente uma linguagem de programação de alto nível para alguma outra forma que viabilize sua execução





Tarefas

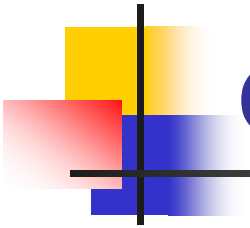
- Deve transformar a L.F. em uma L.A. equivalente
- Deve informar os erros do programa fonte, caso existam
- Permite que os programas sejam independentes de máquina (portabilidade)



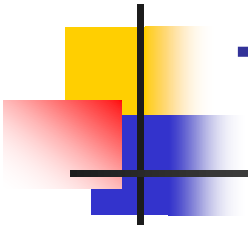
Linguagem de Alto Nível

- Segundo Price & Toscani (2001) as linguagens de programação podem ser classificadas em 5 gerações:
  - 1ª linguagens de máquina
  - 2ª linguagens simbólicas (assembly)
  - 3ª linguagens orientadas ao usuário (linguagens procedimentais e declarativas)
  - 4ª linguagens orientadas à aplicação (projetadas para facilitar a programação por usuários finais)
  - 5ª linguagens de conhecimento (usadas na área de IA)
- As 1ª e 2ª são consideradas de baixo nível e as demais de alto nível.

Propriedades de um bom compilador



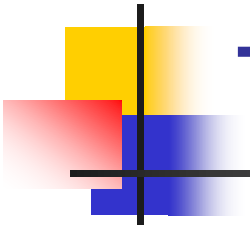
- Principal: gere código corretamente
- Seja capaz de tratar de programas de qualquer tamanho que a memória permita
- Velocidade da compilação não é a característica principal
- Tamanho do compilador já não é mais um problema atualmente
- User-friendliness se mede pela qualidade das mensagens de erros
- A importância da velocidade e tamanho do código gerado depende do propósito do compilador --- velocidade do código vem em primeiro lugar



Tecnologia de Compiladores

- Os primeiros compiladores foram construídos *ad hoc*, sem técnicas, eram vistos como processos complexos e de alto custo

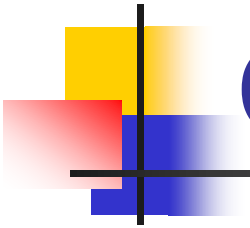
- Hoje, o processo é bem entendido, sendo a construção de um compilador uma rotina, embora a construção de um compilador eficiente e confiável seja ainda uma tarefa complexa.



Tecnologia de Compiladores

- A tecnologia de compiladores é largamente empregada em muitas áreas:
  - linguagens de formatação de textos (nroff, troff, latex, sgml)
  - "silicon compilers" que transformam uma linguagem de especificação de circuitos VLSI em um projeto de circuitos
  - linguagens de acesso a banco de dados (query languages)
  - editores de texto orientados a sintaxe
  - analisadores estáticos de programa que podem descobrir variáveis não inicializadas, código que nunca será executado, etc.
  - checadores de entrada de dados em qualquer sistema
  - interpretadores de comandos de um S.O.

Formas de Classificar Compiladores



- De acordo com o tipo de linguagem objeto gerado

- De acordo com o formato do código objeto gerado

Alternativas para a saída de um compilador (1/4)

- Os compiladores podem ser distinguidos de acordo com o tipo de linguagem objeto que eles geram:
  - Código de Máquina Puro: geram código para um conjunto de instrução de máquina sem assumir a existência de rotinas do SO ou rotinas de biblioteca.

    Isto quer dizer que o compilador não está contando com nenhum "ambiente de execução" 

    Esta abordagem é rara e é utilizada para implementar as linguagens que serão utilizadas para escrever o SO e outros programas de baixo nível.
  - Código de Máquina Aumentado: geração de código para uma arquitetura aumentada com as rotinas do SO (salvar registradores, acessar BIOS) e de suporte à linguagem (I/O, alocação dinâmica, e funções matemáticas) que devem ser carregadas com o programa.

    A combinação de código de máquina do compilador + rotinas do SO + rotinas de suporte é chamada "Máquina Virtual".

    Esta abordagem é mais freqüente.

Alternativas para a saída de um compilador (2/4)

- Código de Máquina Virtual: caso extremo de definição de uma máquina virtual em que o código gerado é composto inteiramente de instruções virtuais: geração para uma máquina hipotética.

  Existe um interpretador que roda sobre o hardware físico e emula o hardware de um projeto diferente.

  Abordagem atrativa pela facilidade de transportar para máquinas diferentes.

  Se a máquina virtual é simples, escrever um interpretador é tarefa fácil (geralmente é a escolha para um "compilador didático")

  Exemplo: "Pascal P-Compiler" 

Alternativas para a saída de um compilador (3/4)

Compiladores também podem ser distinguidos pelo formato do código objeto que eles geram:

- Linguagem Assembly (Montagem): esta abordagem simplifica a tradução pois várias decisões são deixadas para o montador (endereço de jumps, por exemplo).
  - Pode-se checar a saída e verificar a corretude (assembly é mais legível). É útil para cross-compilers (compilador que roda em uma máquina e gera código para outra) pois gera um arquivo facilmente transportável.

  Esta abordagem não é comum, pois precisa de um passo adicional feito pelo montador, que é lento. Algumas vezes o código em assembly é um produto extra de alguns compiladores (psedo-assembly languages) para checagem de corretude.

- Código Binário Relocável: código em que as referências externas e endereços de memória não estão decididos (são endereços simbólicos).

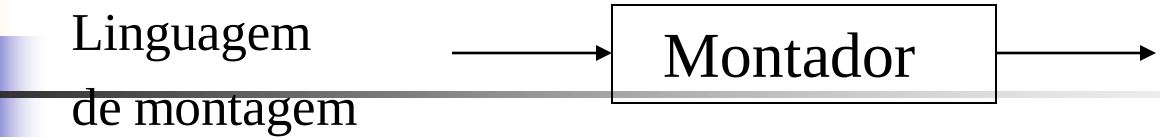
  - Um passo extra de link-edição (link-editor) é necessário para adicionar as rotinas de biblioteca e outras rotinas pré-compiladas (.obj) que são referenciadas dentro do programa compilado, e assim produzir o código binário absoluto. (Permite compilação independente).

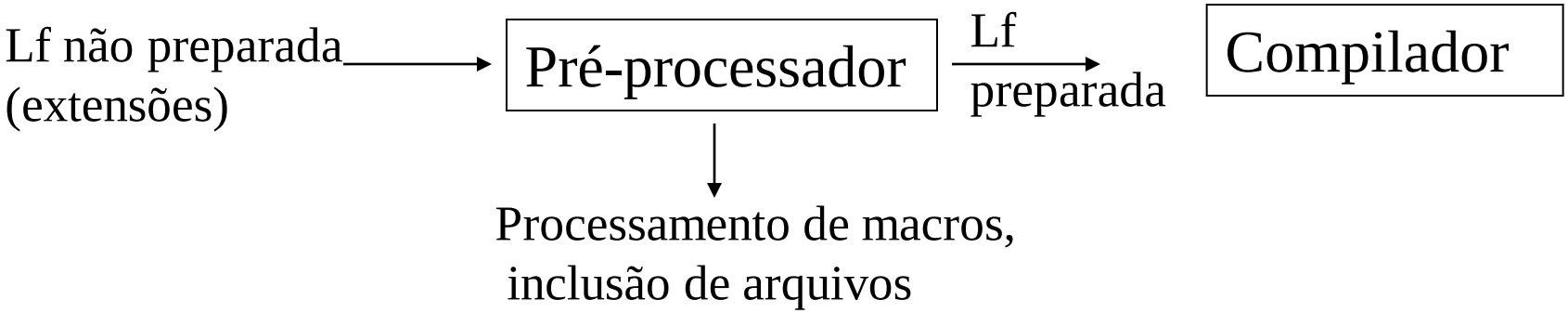
Alternativas para a saída de um compilador (4/4)

O código relocável e o código assembly permitem compilação modular, interfaces com outras linguagens (cross-languages references, isto é, chamadas de rotinas em assembly ou em outras linguagens de alto nível --- Amzi Prolog) e rotinas de biblioteca.

- Imagem de Memória (Load-and-Go): a saída do compilador é carregada diretamente na memória e executada ao invés de ser deixada em um arquivo. Esta abordagem é rápida pois não exige link-edição, mas o programa deve ser recompilado a cada execução. Por ser mais rápido é útil para a fase de depuração e uso de estudantes novatos. Exemplo: Turbo Pascal.

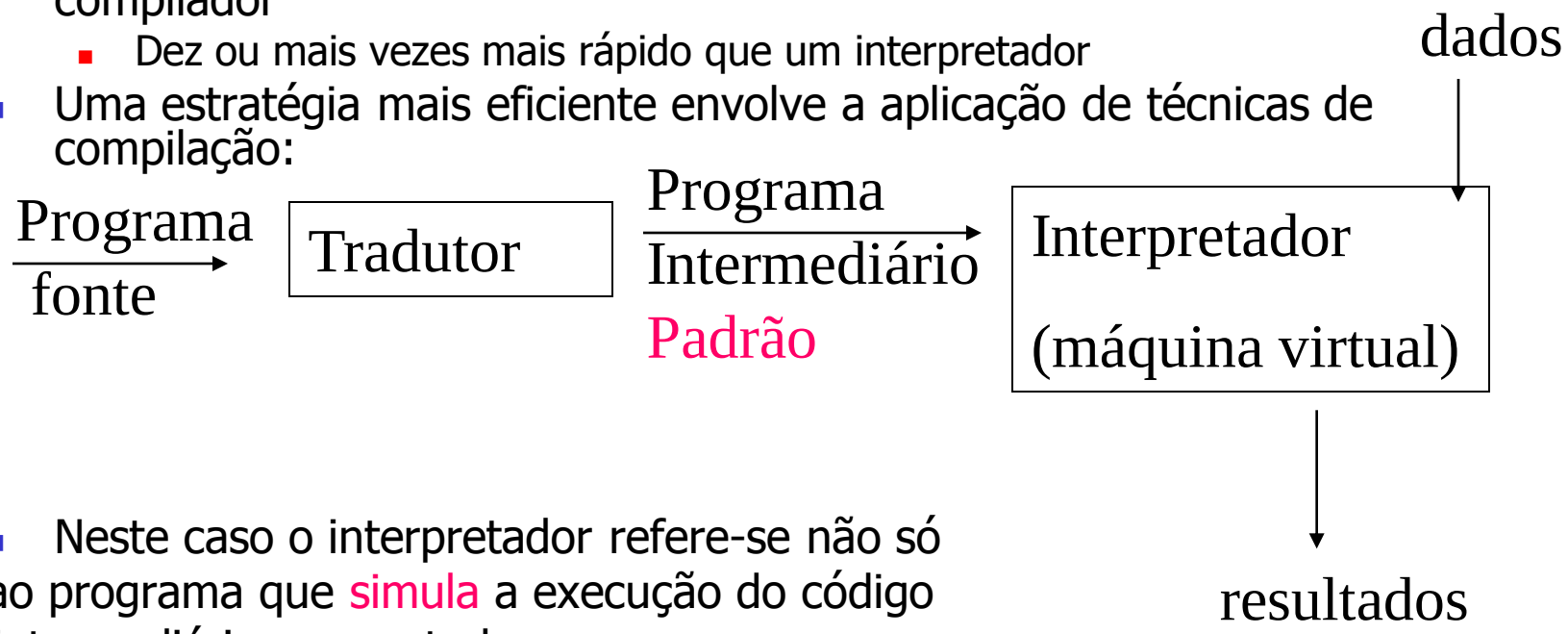
Montadores, Filtros, e Pré-processadores





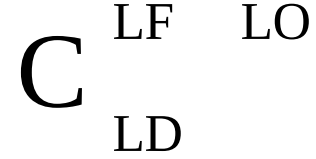
Interpretadores

- Em vez de traduzir de uma vez o programa-fonte para então executá-lo, decodifica unidades básicas do programa para executá-las imediatamente (não mantém o código-objeto para sempre). Este é um enfoque de interpretação (TRADUZ-EXECUTA), porém consome muito tempo. Ex: BASIC.
- Se a velocidade de compilação for requisito importante prefere-se um compilador
  - Dez ou mais vezes mais rápido que um interpretador
- Uma estratégia mais eficiente envolve a aplicação de técnicas de compilação:



- Neste caso o interpretador refere-se não só ao programa que simula a execução do código Intermediário, mas a todo o processo.

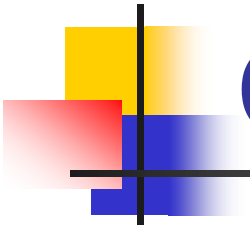
Compilador cruzado, auto-residente, auto-compilável



- Compilador cruzado roda em uma máquina e produz código para outra.
- Compilador auto-residente (ou nativo) é executado na mesma máquina para qual gerou código objeto.

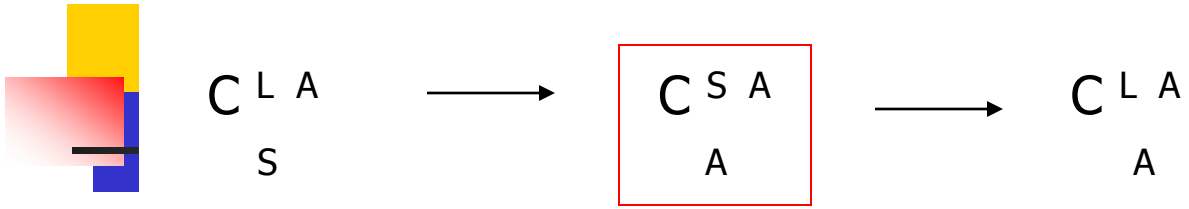
- Compilador auto-compilável. Porque alguém faria um compilador para uma linguagem sendo implementado nela própria????

Compilador auto-compilável



- Lisp foi a primeira linguagem a ter um em 1962. C é outra!
- Há vantagens? Facilita o transporte de uma máquina para outra. Suponha que queremos uma linguagem L para as máquinas A e B.
- 1o. Passo: C $^{S\ A}_{A}$ Pode ser escrito numa linguagem aceita por A, assembly, por exemplo

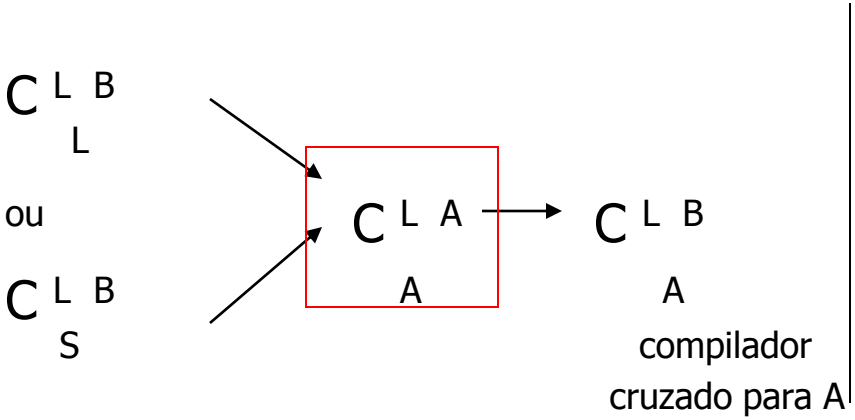
- 2o. Passo: $C^{L\ A}_{S}$

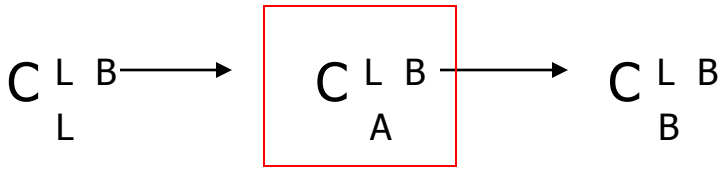


Agora suponha que desejamos ter $C^{L\ B}_{B}$

Como temos $C^{L\ A}_{S}$ fica mais fácil convertê-lo em $C^{L\ B}_{L}$

do que escrever um outro compilador (só temos que mudar a geração de código). Processo em dois passos:

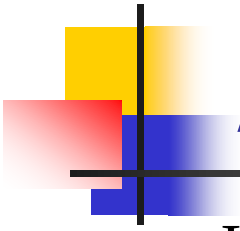




Compiler Compilers

- Ferramentas para auxiliar o desenvolvimento de compiladores:
  - geradores de analisadores léxicos (LEX, por exemplo)
  - geradores de analisadores sintáticos (YACC, por exemplo, possui também chamadas de rotinas para as ações semânticas e geração de código)
  - Geradores de analisadores léxicos e sintáticos integrados como o JavaCC
  - Geradores de código (backend generators) como BEG - Code Generator Generation Tool for Creating Portable Compiler(s)

http://catalog.compilertools.net/



Ambiente de Execução

Infra-estrutura com que o programa objeto conta para sua correta execução:

- uma interface com o SO
- um conjunto de rotinas que deixam disponíveis os recursos oferecidos pela linguagem (rotinas aritméticas, trigonométricas, pacotes gráficos, etc.)
- um conjunto de procedimentos de manutenção do ambiente: gerenciamento de memória, da passagem de parâmetros, do controle da recursividade, etc.

Pascal P-Compiler

- Escrito por um grupo encabeçado por N. Wirth.
- Este compilador gera código chamado P-Code para uma máquina virtual à pilha.
- Problema: a velocidade era 4 vezes menor que o código compilado, mas podemos traduzir o P-Code para um código de máquina real.
- Este método difundiu o Pascal (Pascal UCSD) e o tornou disponível para uma grande variedade de máquinas.
- http://en.wikipedia.org/wiki/P-Code

Stack Machine X Register Machine

- No passado máquinas a pilha reservavam uma parte da memória principal para avaliar expressões. Esta implementação era mais lenta que o uso de registradores pois requer muitas referências à memória principal.
- Máquinas como Lilith, Burroughs e HP usam um pequeno array de registradores rápidos para implementar a pilha de expressões.
- Uma máquina a pilha pode ser mais rápida do que máquinas baseadas em registradores, pois necessitam de poucas instruções.
- Pascal Simplificado permite procedimentos recursivos e seu uso está intimamente relacionado ao uso de pilhas.
- Máquinas a pilha suportam efetivamente linguagens de alto nível estruturadas por blocos (proc. e funções).

Exemplo de Código Binário Relocável

Duas classes de instruções precisam de mais informações:
- referências a endereços de variáveis e funções externas.
- referências a endereços de memória, por exemplo: