

**CENTRO UNIVERSITÁRIO DA FUNDAÇÃO EDUCACIONAL GUAXUPÉ —
UNIFEG**

GABRIEL GIAN FONSECA LEMOS

PORTFÓLIO DE COMPILADORES

**GUAXUPÉ
2021**

Fonseca Lemos, Gabriel Gian
Portfólio de Compiladores / Gabriel Gian Fonseca
Lemos. - Guaxupé, 2021.
8 p. ; 30 cm.

Trabalho - Centro Universitário da Fundação
Eduacional Guaxupé -- UNIFEG, Guaxupé, 2021.

I. UNIFEG

1 INTRODUÇÃO

Na estratificação usual da programação de computadores, as linguagens de programação que utilizamos para instruir os processadores das máquinas a executarem determinado tipo de tarefa tendem a ser categorizadas como linguagens de alto ou baixo nível. O nível em si descreve o grau de estratificação — lê-se: complexidade de implementação crescente e inteligibilidade humana, inversamente, decrescente.

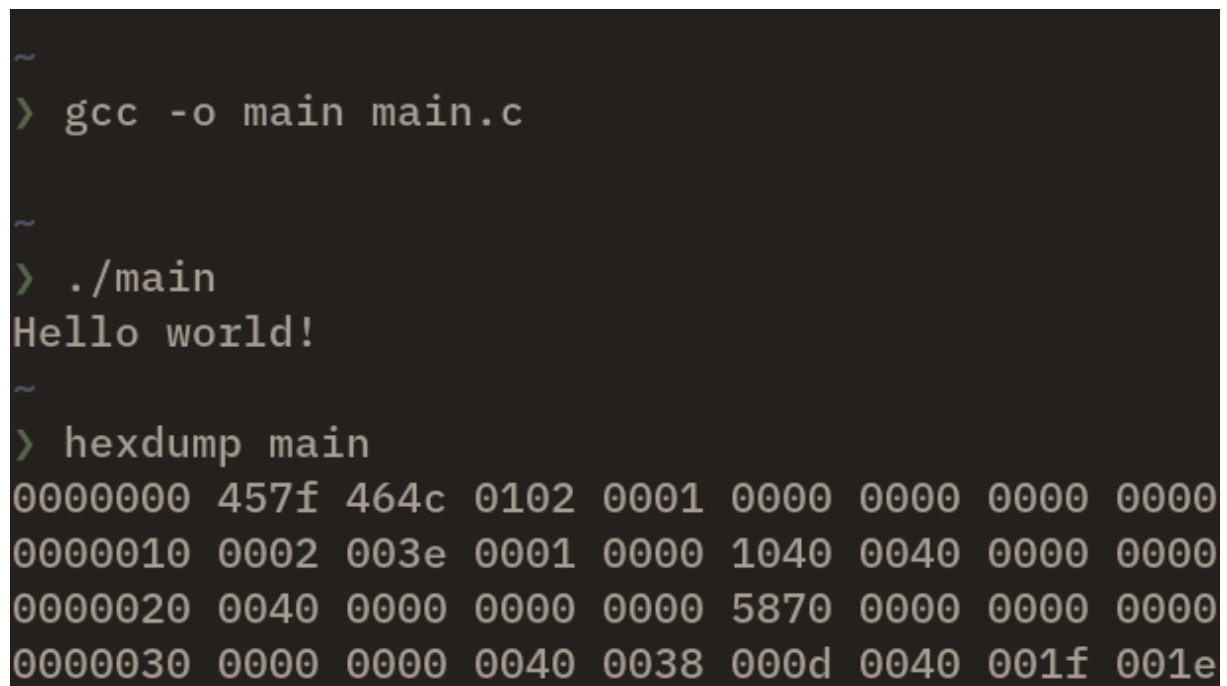
Linguagens de alto nível abstraem, no processo de sua estratificação em estruturas de código mais robustas, conceitos complexos em símbolos simples — expressões de um conteúdo que, para o programador que faz uso destas linguagens, deve estar de certa forma oculto para que o programador possa desenvolver seu software de maneira mais dinâmica, direcionando sua atenção e energia para o problema a ser resolvido, não mais tendo que dividir estes recursos mentais e físicos com as imprescindíveis minúcias do gerenciamento de memória ou dos laços de repetição confusos e ilegíveis no nível do código de máquina.

2 O CÓDIGO DE MÁQUINA

É de senso comum a noção de que computadores leem 0s e 1s, binária. Na prática, o processador recebe instruções do que fazer por meio de um fluxo de bits, divididos em bytes, que, quando traduzidos internamente para uma determinada instrução, fazem com que o processador a execute. Invariavelmente, é óbvio que manter uma tabela de traduções para números binários é demasiadamente difícil e longe do ideal.

Para a facilitação do ato de programar, abstrações foram sendo feitas criando-se representações mais legíveis a seres humanos que descrevam construtos lógicos de programação mais complexos, possibilitando que muito seja feito escrevendo-se pouco.

Linguagens de alto nível tendem a ser escritas em sintaxes amigáveis para ser, logo após, traduzidas em um código otimizado que é de fato lido pelo processador — o processo de compilação. Dividida em vários estágios, a compilação é focada em converter linguagens de alto nível em código de máquina. Para tanto, compiladores costumam implementar também um montador, software capaz de converter o código compilado em uma linguagem de montagem que é uma notação direta do código de máquina sendo executado pelo processador. É a linguagem Assembly. Caso queiramos de fato ler o que está sendo executado pelo processador, uma linguagem de montagem é uma das diversas notações escritas (como por exemplo a representação hexadecimal) do grande número binário que é de fato processado pelo processador.



```
~  
> gcc -o main main.c  
~  
> ./main  
Hello world!  
~  
> hexdump main  
00000000 457f 464c 0102 0001 0000 0000 0000 0000  
00000010 0002 003e 0001 0000 1040 0040 0000 0000  
00000020 0040 0000 0000 0000 5870 0000 0000 0000  
00000030 0000 0000 0040 0038 000d 0040 001f 001e
```

Figura 1 – *Dump* hexadecimal de um programa hello world em C.

2.1 O CÓDIGO EM SI

Uma instrução, na maioria das arquiteturas de processador, pode ser dividida em duas partes principais: o código de operação (operation code ou *opcode*) e o operando, que serve como parâmetro ou dado a ser processado.

```

Conteúdo da secção .debug_line_str:
0000 2f686f6d 652f6765 69737400 6d61696e  /home/geist.main
0010 2e63002f 7573722f 696e636c 75646500  .c./usr/include.
0020 73746469 6f2e6800  stdio.h.

Desmontagem da secção .init:

0000000000401000 <_init>:
401000: f3 0f 1e fa      endbr64
401004: 48 83 ec 08      sub    $0x8,%rsp
401008: 48 8b 05 e9 2f 00 00 mov    0x2fe9(%rip),%rax      # 403ff8 <__gmon_start__>
40100f: 48 85 c0          test   %rax,%rax
401012: 74 02            je     401016 <_init+0x16>
401014: ff d0            callq  *%rax
401016: 48 83 c4 08      add    $0x8,%rsp
40101a: c3              retq

Desmontagem da secção .plt:

0000000000401020 <.plt>:
401020: ff 35 e2 2f 00 00 pushq  0x2fe2(%rip)      # 404008 <_GLOBAL_OFFSET_TABLE_+0x8>
401026: ff 25 e4 2f 00 00 jmpq   *0x2fe4(%rip)      # 404010 <_GLOBAL_OFFSET_TABLE_+0x10>
40102c: 0f 1f 40 00      nopl   0x0(%rax)

0000000000401030 <printf@plt>:
401030: ff 25 e2 2f 00 00 jmpq   *0x2fe2(%rip)      # 404018 <printf@GLIBC_2.2.5>
401036: 68 00 00 00 00 00 pushq  $0x0
40103b: e9 e0 ff ff      jmpq   401020 <.plt>

```

Figura 2 – *Dump* de objetos de um programa hello world em C apresentando o código assembly relacionado.

A emissão das instruções a ser controladas pelo firmware ou pelo circuito integrado da CPU é intermediada pelas chamadas de sistema do sistema operacional em utilização, de forma que, em computadores modernos, a despeito dos arcaicos mainframes dos anos 50 — que eram de fato programados em código de máquina e executados diretamente pelo processador —, os softwares modernos têm seu funcionamento mediado por um sistema operacional — muito provavelmente advindo do Unix —, que isola suas instruções em processos os seus processos em *threads*, e encontra-se ainda mais estratificado.

O código de máquina pode ser gerado tanto pelo processo de compilação — seja um compilador *ahead of time*, que compila todo o código para código de máquina de antemão, antes da execução do mesmo; seja um compilador *just in time*, que compila um código intermediário, em tempo de execução, poupando tempo de compilação ao passo que sacrifica performance — quanto de interpretação, onde, em tempo de execução, o código é traduzido para código de máquina conforme é executado diretamente de uma linguagem de alto nível, muitas vezes fazendo uso de uma máquina virtual ou um software interpretador propriamente dito, um *runtime*.

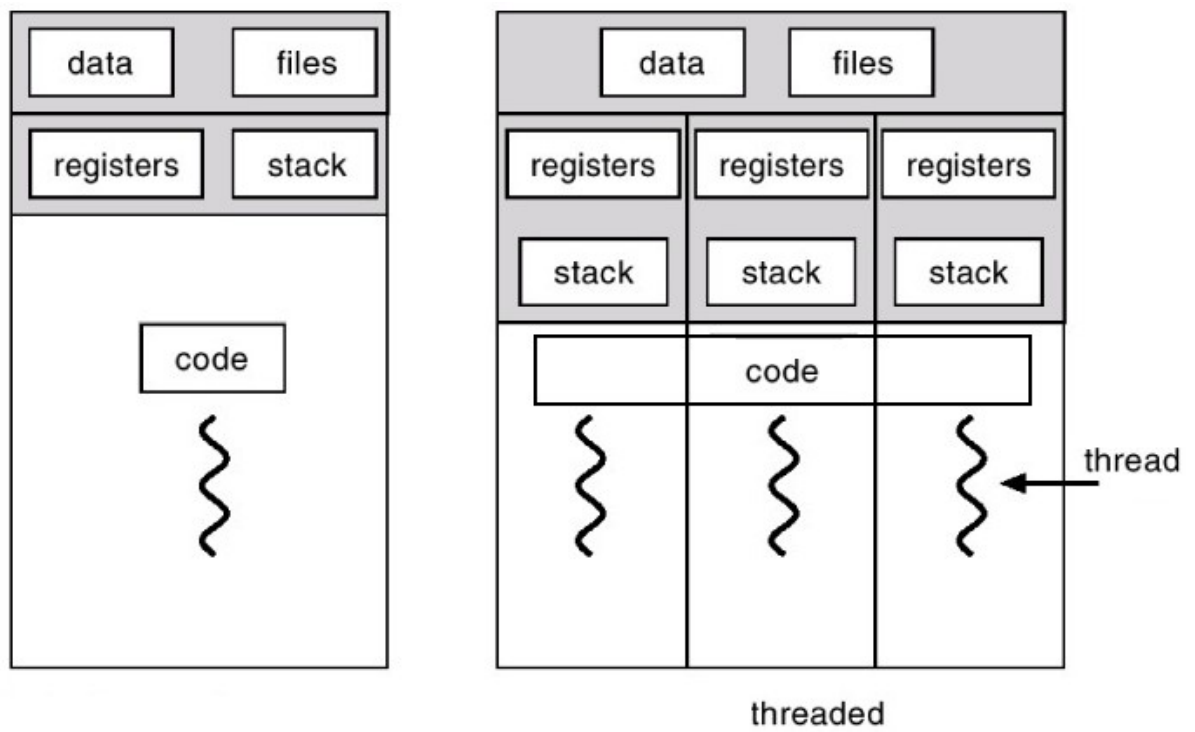


Figura 3 – Esquema de threads em um sistema POSIX.

3 O CÓDIGO DE MÁQUINA, HOJE

Com o advento das linguagens interpretadas (como Python, Ruby e JavaScript) ou executadas em máquinas virtuais (como Erlang, Java e C#), o código de máquina tem se tornado um fato cada vez mais distante do dia a dia do programador médio. A complexidade no tratamento de erros de memória e de otimização, a dificuldade à acessibilidade suscitada pela ininteligibilidade de notações como Assembly no que concerne a construções lógicas mais elaboradas, e as necessidades do mercado em geral têm movido a comunidade de desenvolvimento a soluções de programação de alto nível que abstraem suplícios — muitas vezes vistos como acadêmicos — da arquitetura de computadores e de software, buscando uma maior dinamicidade no desenvolvimento.

Isso não significa que o baixo nível esteja totalmente fora de questão no âmbito do desenvolvimento contemporâneo. Diversos campos da tecnologia, além ou aquém da academia, fazem uso constante de programação em código de máquina, como por exemplo na manufatura de chips de hardware e no desenvolvimento de drivers para os mesmos; na programação de dispositivos em sistemas embarcados que necessitam de performance otimizada e são dotados de poucos recursos; no desenvolvimento do estado da arte, seja da matemática computacional, seja da computação quântica ou até mesmo em cálculos da física de partículas ou astronomia.

4 CONCLUSÃO

O código de máquina, muito mais que uma notação para a descrição de instruções de processadores, é uma *interface* direta entre seres humanos e computadores. Possuir conhecimento acerca da realidade subjacente à linguagem de níveis mais altos permite aos desenvolvedores criar softwares de melhor acurácia lógica, uma vez que a programação em código de máquina requer toda uma recontextualização do modo de pensar muitas vezes relaxado dos desenvolvedores de linguagens de alto nível.

Um conhecimento anagógico da arquitetura de computadores é imprescindível à programação mais precisa e performática de softwares de importância muitas vezes crítica, como sistemas de veículos e aparelhos médicos, assim como à programação de softwares mais supérfluos, uma vez que estes também são capazes de entregar valor — muitas vezes ainda mais que softwares de importância crítica. Com efeito, todo software deve ser tratado como arte.

REFERÊNCIAS

DELEUZE, Gilles; GUATTARI, Félix. 10.000 a.C. — A geologia da moral (quem a Terra pensa que é?). In: _____. (org.). Mil Platôs: Capitalismo e esquizofrenia 2, vol. 1. São Paulo: Editora 34, 2011. p. 69 - 115.

AHO, Alfred; LAM, Monica; SETHI, Ravi; ULLMAN, Jeffrey. Compiladores: Princípios, técnicas e ferramentas. São Paulo: Pearson Addison-Wesley, 2008.

KUO, Tei-Wei. National Taiwan University. UNIX Process Management. 2001. Disponível em: <https://www.csie.ntu.edu.tw/~ktw/InfoSys/PDF_file/UNIX_Process_Management.pdf>. Acesso em: 10 jun. 2021.