ChatGPT

Montadores de Linguagem Assembly: panorama e comparativo

Montadores (ou *assemblers*) são ferramentas que traduzem código em linguagem assembly para código de máquina executável. Eles convertem mnemônicos em opcodes correspondentes, resolvendo endereços de memória, rótulos e inserindo diretivas de montagem para gerar binários ou objetos prontos para execução [1] [2]. Em geral, o montador interpreta sintaxe e macros, gera arquivos objeto (por exemplo, ELF, COFF ou binários planos) e delega à ferramenta de linkedição (*linker*) a resolução final de símbolos e ligações. Montadores podem ser "one-pass" ou "multi-pass", e suportam diferentes formatos de saída, sintaxes (Intel ou AT&T), extensões e recursos de macro, o que afeta diretamente sua aplicabilidade. Por exemplo, NASM e FASM geram diretamente imagens planas (úteis para bootloaders) [3], enquanto GAS produz objetos compatíveis com o *toolchain* GNU (como ELF ou COFF) integrados ao GCC.

Em resumo, montadores permitem programar **diretamente em nível de hardware** (quando se busca controle fino ou otimização máxima) convertendo cada instrução em seus bytes correspondentes. Eles são fundamentais em áreas de baixo nível como inicializadores (*bootloaders*), kernels de sistemas operacionais, firmware e sistemas embarcados, onde são cruciais para desempenho e controle preciso dos recursos de hardware [4] [5]. A tabela abaixo resume alguns montadores importantes:

Montador	Sintaxe padrão	Arquiteturas suportadas	Plataformas	Licença/ Modelo	Uso típico / Notas
NASM (Netwide Assembler)	Intel (fonte→destino) [6]	x86: 16, 32, 64 bits [7]	Linux, Windows, macOS, DOS, etc. [8]	BSD livre [9]	Montador open-source versátil, suporta muitos formatos de objeto (ELF, PE, Mach-O, binário plano) [10]. Usado em código de sistema, bootloaders e kernels por gerar binários diretos [3].

Montador	Sintaxe padrão	Arquiteturas suportadas	Plataformas	Licença/ Modelo	Uso típico / Notas
MASM (Microsoft Macro Assembler)	Intel	x86: 16, 32, 64 bits [11]	Windows, MS-DOS	Proprietário Microsoft (EULA) [12]	Montador fechado da Microsoft, integrado no Visual Studio. Suporta macros avançados e convenções *"alto-nível"* [11] . Predominante no desenvolvimento Windows (drivers, aplicativos legados) – ex.: *RollerCoaster Tycoon* foi escrito 99% em MASM [13] .
GAS (GNU Assembler)	AT&T por padrão (suporta Intel via diretiva) [14] [15]	Multi-arquitetura (x86, ARM, MIPS, RISC-V, SPARC etc.) [16]	Linux/Unix (também Windows via MinGW/ Cygwin)	GNU GPL (binutils, livre)	Montador do *toolchain* GNU. Suporta muitas arquiteturas e formatos (ELF, COFF, Mach-O) [17] [18] . É padrão em sistemas Linux/Unix (incluindo kernel do Linux) [19] e em compilação cruzada para sistemas embarcados.

Montador	Sintaxe padrão	Arquiteturas suportadas	Plataformas	Licença/ Modelo	Uso típico / Notas
FASM (Flat Assembler)	Intel	x86: 32, 64 bits [20]	Windows, Linux, DOS, (portado para outros)	BSD (livre) [21]	Montador open-source escrito em assembly, rápido e *self-hosting*. Foco em velocidade e binários compactos [20]. Gera executáveis próprios (MZ, PE, ELF, binário plano) [22]. Muito usado em projetos de sistemas operacionais independentes (por exemplo, **KolibriOS** foi escrito inteiramente em FASM [23]) e em bootloaders.
YASM	Intel e AT&T (modular, suporta sintaxes NASM/ GAS) [24]	x86: 32, 64 bits	Linux, Windows, macOS	BSD (livre) [25]	Reescrita de NASM sob licença BSD [24], com recursos estendidos (suporta sintaxe de GNU as, gera DWARF, STABS). Compatível com código NASM/ GAS. Usado em bibliotecas que precisam de assembly otimizado cross-platform.

Montador	Sintaxe padrão	Arquiteturas suportadas	Plataformas	Licença/ Modelo	Uso típico / Notas
Outros	–	–	–	–	*TASM* (Borland Turbo Assembler): Intel, MS-DOS/ Windows, obsoleto. *Open Watcom ASM (WASM)*: MASM-compatível, multi-OS, open-source. Assemblers ARM (ARMASM, Keil µVision): usados em desenvolvimento embarcado ARM [26] [27].

Detalhamento dos montadores principais

NASM (Netwide Assembler): é um montador open-source amplamente usado em x86. Suporta código de 16, 32 e 64 bits [7] e produz diversos formatos (ELF, COFF/PE, Mach-O, ou flat binary) [10]. Utiliza sintaxe Intel (fonte→destino) [6]. Suas macros são poderosas, mas mais simples comparadas às de MASM. Pelo seu estilo *standalone* e portabilidade (Linux, Windows, macOS, DOS etc.), NASM é popular em sistemas embarcados e sistemas operacionais experimentais. Em particular, NASM pode gerar imagens planas diretamente usadas em bootloaders e ROMs [3], sendo comum em projetos de inicialização (por exemplo, código de MBR ou *bootstraps* do Linux/GRUB). NASM é distribuído sob licença BSD, sem vínculos proprietários [9].

MASM (Microsoft Macro Assembler): montador proprietário da Microsoft, disponível em Windows e MS-DOS [28]. Suporta IA-32 e x86-64 [11] e adota sintaxe Intel. MASM foi criado em 1981 e hoje é entregue com o Visual Studio/SDKs, focado em desenvolvimento Windows. Suas macros são muito sofisticadas, permitindo lógica complexa (*loops*, expressões aritméticas, pseudo-código de alto nível) [11]. Destina-se sobretudo a software Windows – drivers, aplicativos de baixo nível e código legado em Assembly. Um exemplo famoso de uso intenso do MASM é o jogo *RollerCoaster Tycoon*, escrito 99% em Assembly por Chris Sawyer [13]. Em suma, MASM é escolhido quando se precisa de integração com ferramentas Microsoft e suporte a códigos complexos no universo Windows.

GAS (GNU Assembler): parte do *GNU Binutils* e principal montador do *toolchain* GCC. É gratuito (GPL) e disponível em Unix/Linux/macOS (com ports no Windows). Suporta inúmeras arquiteturas além de x86 – incluindo ARM, MIPS, PowerPC, SPARC, RISC-V etc. [17]. Por padrão usa sintaxe AT&T (destino ← fonte, prefixos % e $), mas aceita sintaxe Intel via diretiva `.intel_syntax` [14] [15]. GAS gera formatos comuns (ELF, a.out, COFF, Mach-O) e integra-se ao compilador C/C++, permitindo *inline asm* e otimizações avançadas. É amplamente empregado em sistemas GNU/Linux: o kernel do Linux e quase todo software baseado em GCC usam GAS para montar trechos em Assembly [19]. Também é muito usado em desenvolvimento embarcado com GCC (ex.: códigos ARM para microcontroladores) [29] [17].

Sua versatilidade o torna onipresente em programação de sistemas, mas a sintaxe AT&T nem sempre é intuitiva para iniciantes acostumados ao estilo Intel.

FASM (Flat Assembler): montador open-source e autocontido, escrito em Assembly por Tomasz Grysztar. Opera em x86 32/64 bits com sintaxe Intel [20]. Destaca-se por ser muito rápido (auto-assemblagem de código) e gerar binários compactos. Suporta macros, porém seu foco é simplicidade e velocidade. FASM produz executáveis prontos (MZ, PE, ELF) e *raw binaries* (incluindo COM) [22]. Como usa poucas opções de linha de comando, desenvolvedores de sistemas operacionais e bootloaders o apreciam pela leveza. De fato, o próprio núcleo e drivers do sistema operacional KolibriOS foram totalmente escritos em FASM [23], aproveitando sua agilidade para criar um sistema minimalista e veloz. FASM também é adotado em projetos de hobby OS e em otimizações que exigem código extremamente enxuto. Ele é distribuído sob licença BSD simplificada (com cláusula de copyleft fraco) [21].

YASM: montador livre (BSD) compatível com NASM e GAS, desenhado para ser modular. Suporta múltiplas sintaxes (NASM, Intel, AT&T) e formatos de objeto (ELF, Mach-O, COFF) [24]. Surgiu em 2001 como reescrita do NASM, ampliando recursos (suporte a x86-64, sintaxe GAS, etc.) [24]. É usado tipicamente em projetos que requerem assembly otimizado em múltiplos sistemas – por exemplo, bibliotecas multimídia (FFmpeg, x264) podem usar YASM nas rotinas críticas de desempenho.

Outros montadores notáveis: *TASM* (Turbo Assembler, da Borland) foi popular em DOS/Windows, sintaxe Intel, mas está descontinuado. *Open Watcom Assembler (WASM)* é um clon do MASM licenciado open-source, usado em projetos legados que migram código DOS/Windows. Para arquiteturas ARM, usam-se assemblers específicos: o **ARMASM** (parte do ARM Development Studio) e o montador do Keil µVision, oferecidos pela Arm Ltd. e por fornecedores de MCU, otimizados para ARM Cortex e amplamente empregados em sistemas embarcados [30] [27]. Em resumo, cada montador foi projetado para certo ecossistema: NASM/FASM para código livre cruzado; MASM/WASM/TASM para ambientes Windows/DOS; GAS para toolchains Unix; assemblers ARM para firmware embarcado; etc.

Sintaxe e compatibilidade

A maioria dos montadores x86 usa **sintaxe Intel** (como NASM, MASM, TASM, FASM, YASM) – isto é, a instrução escreve primeiro o operando-fonte e depois o destino [31]. Já a sintaxe **AT&T** (origem nos sistemas Unix) difere no inverso e em prefixos (% para registradores, $ para literais). O GAS adota AT&T por padrão, mas pode usar Intel com diretiva (ex.: `.intel_syntax noprefix`) [14] [15]. Em arquiteturas, NASM/FASM/YASM/MASM focam x86/IA-32/x86-64, enquanto GAS é retargetável para múltiplas CPUs (ARM, RISC-V, etc.) [17] [14].

As licenças variam: GAS (como parte do GNU) é livre (GPL), NASM e FASM são BSD (livre) [9] [21], MASM é proprietário da Microsoft [12]. Essa diferença legal influencia o uso: montadores livre atraem projetos de código aberto cruzado, já montadores proprietários restringem-se a plataformas específicas. Em termos de comunidade, GAS e NASM têm grandes bases de usuários no mundo Linux/Open Source; MASM e TASM têm histórico em Windows/DOS; FASM/YASM contam com comunidades menores porém dedicadas de entusiastas de baixo nível.

A tabela abaixo resume as principais características técnicas:

Montador	Sintaxe padrão	Arquiteturas	Plataformas	Licença
NASM	Intel (ex.: `mov eax, ebx`) [6]	IA-16/32/64 (x86) [7]	Linux, Windows, macOS, DOS, etc.	BSD livre [9]
MASM	Intel	IA-16/32/64 [11]	Windows, MS-DOS	Microsoft (proprietária) [12]
GAS	AT&T (padrão) / Intel via diretiva [14]	x86, ARM, MIPS, PowerPC, RISC-V, etc. [17]	Linux/Unix (macOS) e Windows (via Cygwin/MinGW)	GNU GPL (livre)
FASM	Intel	IA-32/64 [20]	Windows, Linux, DOS, KolibriOS, etc.	BSD (livre) [21]
YASM	Intel / AT&T	IA-32/64 (como NASM)	Multi-plataforma (via GNU GCC)	BSD (livre) [25]
TASM/ WASM	Intel	IA-16/32 (TASM), x86/x64 (WASM)	DOS/Windows	Proprietária (TASM); Livre Sybase (WASM)

Cada caso de uso valoriza diferentes atributos. Por exemplo, NASM e FASM são preferidos em **desenvolvimento de sistemas operacionais e bootloaders** por gerar binários diretos e permitirem controle total do código. [3] [23] Já MASM é usado em **aplicações Windows** e drivers de baixo nível devido à sua integração com o ecossistema Microsoft [11]. GAS domina em **Unix/Linux e sistemas embarcados** por estar no GCC, formando a base de compilações cruzadas e do próprio kernel do Linux [19] [29]. Em **otimização de desempenho**, qualquer montador pode ser usado, mas projetos multiplataforma costumam escolher NASM/GAS/YASM, enquanto código específico de Windows usaria MASM.

Exemplos de uso típicos

- **Sistemas operacionais e bootloaders:** Projetos de *OSDev* costumam usar NASM ou FASM. Por exemplo, muitos *bootloaders* x86 (como o código inicial do GRUB) e kernels escritos do zero são montados com NASM [3]. O **KolibriOS** (pequeno SO experimental) teve seu kernel e drivers totalmente implementados em FASM [23], explorando sua velocidade. Em contraponto, o kernel Linux é montado com GAS [19] e seus drivers de dispositivo também usam GAS (integrado ao GCC).
- **Desenvolvimento embarcado:** Em sistemas ARM e microcontroladores, geralmente usa-se o assembler nativo do fabricante. Por exemplo, o **ARMASM** (da ARM Ltd.) e o assembler do Keil µVision são comuns em projetos Cortex-M [30] [27]. Na linha GNU, GAS é usado para cross-compilação ARM (via `arm-none-eabi-as`), pois suporta arquiteturas ARM e produz ELF para firmware [29] [17].
- **Windows e drivers:** No mundo Windows, MASM é quase padrão. Bibliotecas gráficas e drivers WDM/WMI podem ter código em MASM ou em drivers de videogame antigos. Um caso famoso foi o jogo *RollerCoaster Tycoon*, quase todo escrito em MASM [13]. Programas antigos de DOS/ Win95 também usavam TASM ou MASM.
- **Otimizações e multimídia:** Bibliotecas que fazem *fast paths* em assembly (multimídia, criptografia) muitas vezes têm versões em NASM/GAS/YASM para x86. Por exemplo, *FFmpeg* utiliza YASM para rotinas de processamento de vídeo em x86/x64 para melhor desempenho. Em geral, desenvolvedores escolhem o montador que melhor se integra ao seu processo de build:

no Linux/GCC, usa-se GAS (ou NASM via `-f elf`); no Windows/Visual Studio, MASM; em builds portáveis, NASM ou YASM.

Em resumo, não existe um **melhor absoluto**: cada montador brilha em um contexto. NASM e FASM destacam-se por portabilidade e leveza [3] [23]; MASM por integração no ecossistema Windows [32]; GAS pela versatilidade multi-arquitetura e presença no mundo Unix [17] [19]. A escolha depende da arquitetura alvo, sistema operacional e necessidades de licenciamento de cada projeto.

Fontes: Documentação de montadores (NASM, MASM, GAS, FASM) e análises comparativas [33] [14] [29] [32] [23]. Cada citação segue o formato `【cursor†Lx-Ly】` para referência precisa.

---

[1] Linguagem assembly – Wikipédia, a enciclopédia livre
https://pt.wikipedia.org/wiki/Linguagem_assembly

[2] [26] [27] [30] Popular Assemblers for x86 and ARM Architectures
https://simplifycpp.org/?id=a0149

[3] [6] [7] [8] [9] [10] [33] Netwide Assembler - Wikipedia
https://en.wikipedia.org/wiki/Netwide_Assembler

[4] [14] [31] Linguagem de montagem x86 – Wikipédia, a enciclopédia livre
https://pt.wikipedia.org/wiki/Linguagem_de_montagem_x86

[5] Comparison of assemblers
https://grokipedia.com/page/Comparison_of_assemblers

[11] [12] [13] [28] [32] Microsoft Macro Assembler – Wikipédia, a enciclopédia livre
https://pt.wikipedia.org/wiki/Microsoft_Macro_Assembler

[15] [16] [17] [18] [29] #2 Mastering GAS A Complete Guide to the GNU Assembler
https://simplifycpp.org/?id=a0844

[19] Which assembly is used by Linux Kernel? Is it really NASM? - Stack Overflow
https://stackoverflow.com/questions/59669393/which-assembly-is-used-by-linux-kernel-is-it-really-nasm

[20] [21] [22] FASM - Wikipedia
https://en.wikipedia.org/wiki/FASM

[23] KolibriOS
https://kolibrios.org/en

[24] [25] Yasm User Manual
https://www.tortall.net/projects/yasm/manual/html/manual.html