WIKIPEDIA

Source-to-source compiler

A **source-to-source translator**, **source-to-source compiler** (**S2S compiler**), **transcompiler**, or **transpiler**[1] is a type of translator that takes the source code of a program written in a programming language as its input and produces an equivalent source code in the same or a different programming language. A source-to-source translator converts between programming languages that operate at approximately the same level of abstraction, while a traditional compiler translates from a higher level programming language to a lower level programming language. For example, a source-to-source translator may perform a translation of a program from Python to JavaScript, while a traditional compiler translates from a language like C to assembler or Java to bytecode.[2] An automatic parallelizing compiler will frequently take in a high level language program as an input and then transform the code and annotate it with parallel code annotations (e.g., OpenMP) or language constructs (e.g. Fortran's `forall` statements).[3]

Another purpose of source-to-source-compiling is translating legacy code to use the next version of the underlying programming language or an API that breaks backward compatibility. It will perform automatic code refactoring which is useful when the programs to refactor are outside the control of the original implementer (for example, converting programs from Python 2 to Python 3, or converting programs from an old API to the new API) or when the size of the program makes it impractical or time-consuming to refactor it by hand.

Transcompilers may either keep translated code structure as close to the source code as possible to ease development and debugging of the original source code, or may change the structure of the original code so much that the translated code does not look like the source code.[4] There are also debugging utilities that map the transcompiled source code back to the original code; for example, the JavaScript Source Map standard allows mapping of the JavaScript code executed by a web browser back to the original source when the JavaScript code was, for example, minified or produced by a transcompiled-to-JavaScript language.

Examples include Closure Compiler, CoffeeScript, Dart, Haxe, TypeScript and Emscripten.[5]

Contents

Assembly language translators
- Intel CONV86
- SCP TRANS86
- Sorcim TRANS86
- Digital Research XLT86
- Others

Programming language implementations

Porting a codebase

Transcompiler pipelines
- Recursive transcompiling

See also

Notes

References

Further reading
External links

Assembly language translators

Intel CONV86

Intel marketed their 16-bit processor 8086 to be source compatible to the 8080, an 8-bit processor.[6] To support this, Intel had an ISIS-II-based translator from 8080 to 8086 source code named CONV86[7][8][9] (also referred to as CONV-86[10] and CONVERT 86[11][12]) available to OEM customers since 1978, possibly the earliest program of this kind.[nb 1] It supported multiple levels of translation and ran at 2 MHz on an Intel Microprocessor Development System MDS-800 with 8-inch floppy drives. According to user reports, it did not work very reliably.[13][14]

SCP TRANS86

Seattle Computer Products' (SCP) offered TRANS86.COM,[12][15][16] written by Tim Paterson in 1980 while developing 86-DOS.[17][18][19] The utility could translate Intel 8080 and Zilog Z80 assembly source code (with Zilog/Mostek mnemonics) into .ASM source code for the Intel 8086 (in a format only compatible with SCP's cross-assembler ASM86 for CP/M-80), but supported only a subset of opcodes, registers and modes, and often still required significant manual correction and rework afterwards.[20][17] Also, performing only a mere transliteration,[11][15][7][8] the brute-force single-pass translator did not carry out any register and jump optimizations.[21][22] It took about 24 KB of RAM.[12] The SCP version 1 of TRANS86.COM ran on Z80-based systems.[12][15] Once 86-DOS was running, Paterson utilized TRANS86 to convert itself into a program running under 86-DOS.[19] Numbered version 2, this was named TRANS.COM instead.[15][22][21][23][24] Later in 1982, the translator was apparently also available from Microsoft.[12][25]

Sorcim TRANS86

Also named TRANS86, Sorcim offered an 8080 to 8086 translator as well since December 1980.[26][11] Like SCP's program it was designed to port CP/M-80 application code (in ASM, MAC, RMAC or ACT80 assembly format) to MS-DOS (in a format compatible with ACT86).[26][12][27][28] In ACT80 format it also supported a few Z80 mnemonics. The translation occurred on an instruction-by-instruction basis with some optimization applied to conditional jumps. The program ran under CP/M-80, MP/M-80 and Cromemco DOS with a minimum of 24 KB of RAM, and had no restrictions on the source file size.[12][29]

Digital Research XLT86

Much more sophisticated and the first to introduce optimizing compiler technologies into the source translation process was Digital Research's XLT86 1.0 in September 1981. XLT86 1.1 was available by April 1982.[30] The program was written by Gary Kildall[11][31][32][33] and translated .ASM source code for the Intel 8080 processor (in a format compatible with ASM, MAC or RMAC assemblers) into .A86 source code for the 8086 (compatible with ASM86). Using global data flow analysis on 8080 register usage,[34][11][35][36] the five-phase multi-pass translator would also optimize the output for code size and take care of calling conventions (CP/M-80 BDOS calls were mapped into BDOS calls for CP/M-86), so that CP/M-80 and MP/M-80 programs could be ported to the CP/M-86 and MP/M-86 platforms automatically. XLT86.COM itself was written in PL/I-80 for CP/M-80 platforms.[37][12][30][38] The program occupied 30 KB of RAM for itself plus additional memory for the program graph. On a 64 KB memory system, the maximum source file size supported was about 6 KB,[37][12][39][30] so that larger files had to be broken down accordingly before translation.[12][30] Alternatively, XLT86 was also available for DEC VMS (for VAX 11/750 or 11/780).[12][30] Although XLT86's input and output worked on source-code level, the translator's in-memory representation of the program and the applied code optimizing technologies set the foundation to binary recompilation.[40][41][42]

Others

2500 AD Software offered an 8080 to 8086 source-code translator as part of their XASM suite for CP/M-80 machines with Z80 as well as for Zilog ZEUS and Olivetti PCOS systems.[43]

Since 1979, Zilog offered a Z80 to Z8000 translator as part of their PDS 8000 development system.[44][45][46][47][48][14] Advanced Micro Computers (AMC)[48][14] and 2500 AD Software offered Z80 to Z8000 translators as well.[43] The latter was named TRANS[49][50] and was available for Z80 CP/M, CP/M-86, MS-DOS and PCOS.[43]

Programming language implementations

The first implementations of some programming languages started as transcompilers, and the default implementation for some of those languages are still transcompilers. In addition to the table below, a CoffeeScript maintainer provides a list of languages that compile to JavaScript.[51]

List of transcompilers[2]

Name	Source language	Target language
Cfront	C++	C
HipHop for PHP (HPHPc)	PHP	C++
Babel	ES6+ (JS)	ES5
ClojureScript	Clojure	JavaScript
JSweet[52]	Java	TypeScript
Swiftify[53]	Objective-C	Swift
J2ObjC[54]	Java	Objective-C
Haxe	Haxe	ActionScript 3, JavaScript, Java, C++, C#, PHP, Python, Lua
Maia[55]	Maia	Verilog
Cerberus X (https://www.cerberus-x.com/community/portal/)	Cerberus	JavaScript, Java, C++, C#

Porting a codebase

When developers want to switch to a different language while retaining most of an existing codebase, it might be better to use a transcompiler compared to rewriting the whole software by hand. Depending on the quality of the transcompiler, the code may or may not need manual intervention in order to work properly. This is different from "transcompiled languages" where the specifications demand that the output source code always works without modification. All transcompilers used to port a codebase will expect manual adjustment of the output source code if there is a need to achieve maximum code quality in terms of readability and platform convention.

Tool	Source language	Target language	Comments
2to3 script	Python 2	Python 3	Even though 2to3 does its best at automating the translation process, further manual corrections are often needed.
Emscripten	LLVM bytecode	JavaScript	This allows running C/C++ codebases in a browser for example
c2go[56]	C	Go	Before the 1.5 release, the Go compiler was written in C. An automatic translator was developed to automatically convert the compiler codebase from C into Go.[57][58] Since Go 1.5, the "compiler and runtime are now implemented in Go and assembler, without C".
C2Rust[59]	C	Rust	C2Rust takes C code as input and outputs `unsafe` Rust code, focusing on preserving compatibility with the original codebase. Several documented limitations exist for this process. Converting the resulting code to safe and idiomatic Rust code is a manual effort post translation, although an automated tool exists to ease this task. [59]
Google Web Toolkit	Java program that uses a specific API	JavaScript	The Java code is a little bit constrained compared to normal Java code.
Js_of_ocaml[60] of Ocsigen	OCaml	JavaScript	
J2Eif[61]	Java	Eiffel	The resulting Eiffel code has classes and structures similar to the Java program but following Eiffel syntax and conventions.
C2Eif[62]	C	Eiffel	The resulting Eiffel code has classes and structures that try to be as clean as possible. The tool is complete and relies on embedding the C and assembly code if it cannot translate it properly.
Swiftify[63]	Objective-C	Swift	Swiftify is an online source to source conversion tool from Objective-C into Swift. It assists developers who are migrating all or part of their iOS codebase into Swift. The conversion is aimed primarily at converting the syntax between Objective-C and Swift, and is helped because Apple took efforts to ensure compatibility between Swift and Objective-C runtimes.
Runtime Converter[64]	PHP	Java	The Runtime Converter is an automatic tool which converts PHP source code into Java source code. There is a Java runtime library for certain features of the PHP language, as well as the ability to call into the PHP binary itself using JNI for PHP standard library and extension function calls.

Transcompiler pipelines

A transcompiler pipeline is what results from *recursive transcompiling*. By stringing together multiple layers of tech, with a transcompile step between each layer, technology can be repeatedly transformed, effectively creating a distributed language independent specification.

XSLT is a general purpose transform tool which can be used between many different technologies, to create such a derivative code pipeline.

Recursive transcompiling

Recursive transpiling (or **recursive transcompiling**) is the process of applying the notion of transpiling recursively, to create a pipeline of transformations (often starting from a single source of truth) which repeatedly turn one technology into another.

By repeating this process, one can turn A → B → C → D → E → F and then back into A(v2). Some information will be preserved through this pipeline, from A → A(v2), and that information (at an abstract level) demonstrates what each of the components A–F agree on.

In each of the different versions that the transcompiler pipeline produces, that information is preserved. It might take on many different shapes and sizes, but by the time it comes back to A (v2), having been transcompiled 6 times in the pipeline above, the information returns to its original state.

This information which survives the transform through each format, from A–F–A(v2), is (by definition) derivative content or derivative code.

Recursive transpiling takes advantage of the fact that transpilers may either keep translated code as close to the source code as possible to ease development and debugging of the original source code, or else they may change the structure of the original code so much, that the translated code does not look like the source code. There are also debugging utilities that map the transpiled source code back to the original code; for example, JavaScript source maps allow mapping of the JavaScript code executed by a web browser back to the original source in a transpiled-to-JavaScript language.

See also

- Binary recompiler
- C to HDL
- Code generation (compiler) – Process by which a compiler's code generator converts some intermediate representation of source code into a form that can be readily executed by a machine
- DMS Software Reengineering Toolkit – a source-to-source compiler framework using explicit pattern-directed rewrite rules
- f2c – a source-to-source compiler from Fortran 77 to C
- Honeywell Liberator (running IBM 1401 programs on Honeywell H200)
- Intermediate representation
- Language binding – Software library that allows using another library coded in another programming language
- Language-independent specification – Computer programming standard meant to be interoperable across programming languages
- Language interoperability
- Object code optimizer, also known as Binary optimization
- Preprocessor – Program which processes the input for another program
- Program transformation
- Recursive transcompiling
- ROSE (compiler framework) – a source-to-source compiler framework
- Translator (computing) – Computer program that translates code from one programming language to another
- XSLT – Language for transforming XML documents

Notes

1. One commercial program known to have been machine-translated under ISIS-II from 8080 CP/M-80 source code to 8086 CP/M-86 using Intel's CONV86 was MicroPro's WordStar 3.0 in September 1981.

References

1. ARC-Softwaresystems (June 1988). "Aus BASIC mach C: B→C Transpiler" (https://archive.org/details/AmigaMagazin198806) [Turn BASIC into C: B→C Transpiler]. *Amiga-Magazin - das Computermagazin für Amiga-Fans* (Advertisement) (in German). Vol. 1988 no. 6. Esslingen, Germany: Markt & Technik Verlag Aktiengesellschaft. p. 101. ISSN 0933-8713 (https://www.worldcat.org/issn/0933-8713). Archived (https://web.archive.org/web/20200201131207/https://archive.org/details/AmigaMagazin198806) from the original on 2020-02-01. Retrieved 2020-01-18. "[…] Achtung C- und Basic-Programmierer! […] Jetzt gibt es den B→C TRANSPILER das einzigartige Umwandlungs-Software-System von ARC […] Der B→C TRANSPILER übersetzt lauffähige AMIGA-Basicprogramme in compilierbaren C-Code. […] Durch Spezialbefehle kann C-Code in Basicproqramme direkt integriert werden. […] Basic-Befehle werden erweitert transpiliert. (HAM-Modus, IFF, usw. werden unterstützt). […] Mit diesem Konzept neuester Generation verbindet der B→C TRANSPILER auf einzigartige Weise die Vorteile eines Interpreters mit denen eines Compilers […]" [1] (https://archive.org/details/AmigaMagazin198806)
2. "Transpiler" (https://devopedia.org/transpiler). *devopedia.org*. Archived (https://web.archive.org/web/20191105090612/https://devopedia.org/transpiler) from the original on 2019-11-05. Retrieved 2019-06-22.
3. "Types of compilers" (http://www.compilers.net/paedia/compiler/index.htm). compilers.net. 1997–2005. Archived (https://web.archive.org/web/20190719090932/http://www.compilers.net/paedia/compiler/index.htm) from the original on 2019-07-19. Retrieved 2010-10-28.
4. Fowler, Martin (2013-02-12). "Transparent Compilation" (http://martinfowler.com/bliki/TransparentCompilation.html). Archived (https://web.archive.org/web/20200101165423/https://martinfowler.com/bliki/TransparentCompilation.html) from the original on 2020-01-01. Retrieved 2013-02-13.
5. Epic Games; Mozilla. "HTML5 Epic Citadel" (http://www.unrealengine.com/html5).
6. Scanlon, Leo J. (1988). *8086/8088/80286 assembly language* (https://archive.org/details/8086808880286ass0000scan/page/12). Brady Books. p. 12 (https://archive.org/details/8086808880286ass0000scan/page/12). ISBN 978-0-13-246919-7. "[…] The 8086 is software-compatible with the 8080 at the assembly-language level. […]"
7. *MCS-86 Assembly Language Converter Operating Instructions For ISIS-II Users* (https://archive.org/details/bitsavers_intelISISIblyLanguageConverterOperatingInstruction_3712591). A30/379/10K TL. Santa Clara, California, USA: Intel Corporation. March 1979 [1978]. Order No. 9800642A. Retrieved 2020-01-18. [2] (https://archive.org/details/bitsavers_intelISISIblyLanguageConverterOperatingInstruction_3712591) (NB. A newer version of this manual can be found here.)
8. *MCS-86 Assembly Language Converter Operating Instructions For ISIS-II Users* (https://archive.org/details/bitsavers_intelISISImblyLanguageConverterFeb80_2443878). A175/280/7.5 FL. Santa Clara, California, USA: Intel Corporation. February 1980 [1978]. Order No. 9800642-02. Retrieved 2020-01-18. [3] (https://archive.org/details/bitsavers_intelISISImblyLanguageConverterFeb80_2443878)[4] (http://www.nj7p.info/Manuals/PDFs/Intel/9800642B.pdf) (NB. An older version of this manual can be found here.)

9. Nelson, Ross P. (January 1989) [1988]. *The 80386 Book: Assembly Language Programmer's Guide for the 80386*. Microsoft Programming Series (1 ed.). Microsoft Press. p. 2. ISBN 978-1-55615-138-5. "[…] An Intel translator program could convert 8080 assembler programs into 8086 assembler programs […]"
10. *The 8086 Family User's Manual* (https://archive.org/details/bitsavers_intel80869lyUsersManualOct79_62967963). Intel Corporation. October 1979 [1978]. pp. 2-74, 2-92, B-176. Order No. 9800722-03. Retrieved 2020-01-18. "[…] other programs round out the software development tools available for the 8086 and 8088. […] CONV-86 can do most of the conversion work required to translate 8080/8085 assembly language source modules into ASM-86 source modules. […] To facilitate conversion of 8080A/8085A assembly language programs to run on the iSBC 86/12A board CONV-86 is available under the ISIS-II operating system." [5] (https://archive.org/details/bitsavers_intel80869lyUsersManualOct79_62967963)[6] (http://bitsavers.informatik.uni-stuttgart.de/components/intel/8086/9800722-03_The_8086_Family_Users_Manual_Oct79.pdf)
11. Freiberger, Paul (1981-10-19). "Program translators do it literally - and sometimes in context" (https://books.google.com/books?id=Gj0EAAAAMBAJ&pg=PA19). *InfoWorld - News For Microcomputer Users*. Special section: Computer compatibility. **3** (22). Popular Computing, Inc. p. 19. ISSN 0199-6649 (https://www.worldcat.org/issn/0199-6649). Archived (https://web.archive.org/web/20200201160014/https://books.google.com/books?id=Gj0EAAAAMBAJ&pg=PA19&lpg=PA19) from the original on 2020-02-01. Retrieved 2020-01-15. "[…] "Unless you have a translating scheme that takes account of the peculiar idiosyncrasies of the target microprocessor, there is no way that an automatic translator can work," explains Daniel Davis, a programmer with Digital Research. "You'll end up with direct transliterations." […] In spite of all these limitations, progress has been made recently in the development of translators. Most notably, Digital Research has introduced its eight- to 16-bit assembly code translator. Based on research performed by Digital Research president Gary Kildall, the XLT86 appears to offer advances over previously available software translator technology. Like Sorcim's Trans and Intel's Convert 86, Kildall's package translates assembly-language code from an 8080 microprocessor to an 8086. However, Kildall has applied a global flow analysis technique that takes into account some of the major drawbacks of other translators. The procedure analyzes the register and flag usage in sections of 8080 code in order to eliminate nonessential code. According to Digital Research programmer Davis, the algorithm Kildall uses allows the translator to consider the context as it translates the program. Until now, one of the major problems with any translator program has been the inability of the software to do much more than transliteration. If Digital Research's new translator actually advances the technology to the point where context can be considered, then more software translators may proliferate in the microcomputer marketplace."

12. Taylor, Roger; Lemmons, Phil (June 1982). "Upward migration - Part 1: Translators - Using translation programs to move CP/M-86 programs to CP/M and MS-DOS" (https://tech-insider.org/personal-computers/research/acrobat/8206-b.pdf) (PDF). *BYTE*. Vol. 7 no. 6. BYTE Publications Inc. pp. 321–322, 324, 326, 328, 330, 332, 334, 336, 338, 340, 342, 344. ISSN 0360-5280 (https://www.worldcat.org/issn/0360-5280). CODEN BYTEDJ (https://cassi.cas.org/searching.jsp?searchIn=codens&exactMatch=y&c=WIy460-R_DY&searchFor=BYTEDJ). Archived (https://web.archive.org/web/20200116024623/https://tech-insider.org/personal-computers/research/acrobat/8206-b.pdf) (PDF) from the original on 2020-01-16. Retrieved 2020-01-15. "[…] Digital Research's XLT86 takes standard 8080 source code in a format compatible with ASM, MAC, or RMAC assemblers and converts the 8080 source code to 8086 source code in a format compatible with ASM86 operating under either CP/M-80 or CP/M-86. Since XLT86 is written in PL/I-80, the translator can run either stand-alone under CP/M-80 or for cross development under VAX/VMS. It produces optimized 8086 code in a five-phase, multipass process, doing global data-flow analysis to determine optimal register usage. Although macro definitions are not supported, conditional-assembly directives are […] if you want macro expansion, you can use a pass through MAC or RMAC to produce a PRN file that can be edited […] to produce an expanded source file for input acceptable to XLT86. XLT86 does not recognize Z80 instructions. XLT86 passes repeat loops through to the 8086 source code. XLT86 analyzes the source program in its entirety, determining the block structure and the register/flag usage. Working from this information, it translates the code to 8086 assembler code in an optimized way. The decision algorithm for each instruction type is given in […] the manual […] Register mapping generally follows […] with a loose relationship between the 8086 AX and the 8080 PSW; the exact relationship is determined from register usage at translate time. Many runtime options are available to control the translation process, both on the command line and embedded in the 8080 source text. […] XLT86 is a sophisticated program that does a reasonable job of optimizing the translation of 8080 source code to 8086 source code. BDOS calls from CP/M-80 are mapped into BDOS calls that are compatible with CP/M-86. XLT86 has special features for handling translation of conditional JMP and CALL instructions in 8080 source code. In the 8080 instructions, JMP and CALL instructions are capable of reaching any address within the 64K-byte region. The 8086 conditional JMP instructions can reach only 128 bytes on either side of the IP […] register. XLT86 examines the target of the conditional JMP. If the target cannot be reached, XLT86 changes the sense of the conditional JMP and skips over a long JMP to the target address. Since there are no conditional CALL or RET instructions in the 8086, the sense of the condition is changed and a short conditional JMP is performed to skip over an unconditional CALL or RET. […] the segment registers allow for separation of code and data regions. […] XLT86 examines an expression and determines the proper segment for the particular instruction. […]" [7] (https://archive.org/details/byte-magazine-1982-06) [8] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0323.pdf)[9] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0324.pdf)[10] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0326.pdf)[11] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0328.pdf) [12] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0330.pdf)[13] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0332.pdf)[14] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0334.pdf)[15] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0336.pdf)[16] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0338.pdf) [17] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0340.pdf)[18] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0342.pdf)[19] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-O

CR-Page-0344.pdf)[20] (https://www.americanradiohistory.com/hd2/IDX-Consumer/Archive-Byte-IDX/IDX/80s/82-83/Byte-1982-06-OCR-Page-0346.pdf) (13 pages)

13. Guzis, Charles "Chuck" P. (2013-02-24). "Re: What if IBM didn't choose the Intel CPU!" (http://www.vcfed.org/forum/archive/index.php/t-35844.html). *Vintage Computer Forum*. Genre: Others. Archived (https://web.archive.org/web/20200116042606/http://www.vcfed.org/forum/archive/index.php/t-35844.html) from the original on 2020-01-16. Retrieved 2020-01-15. "[…] The original 8086 assembler ran on an 8080-equipped MDS-80 dev system. One of the first products was a 8080 to 8086 source level translator. I recall that the speed of translation was phenomenally slow. […] our sales guy offered to run a conversion and verification test at the local sales office […] We […] started the job on the ISIS-II MDS 200 series there--they even had a hard disk, which was an outrageously expensive option for an MDS […] still crunching when we left for the night. It still wasn't done the next morning […] About 2 weeks later, after the Intel software guys had a look at the translator, Ed returned with the translated program. It was about 50% larger in size than the original 8085 version, which sort of went counter to Intel's claims for the translator. […]"
14. Guzis, Charles "Chuck" P. (2016-12-31) [2016-12-30]. "Re: DOS code in CP/M? Revisited…" (http://www.vcfed.org/forum/archive/index.php/t-53239.html). *Vintage Computer Forum*. Genre: CP/M and MP/M. Archived (https://web.archive.org/web/20200116032911/http://www.vcfed.org/forum/archive/index.php/t-53239.html) from the original on 2020-01-16. Retrieved 2020-01-15. "[…] Intel had an ISIS-hosted translator from 8080-to-8086 code. I can remember spending a very frustrating day at the local Intel sales office with a sample bit of 8080 code--a basic BCD floating-point package for the 8080 and waiting for hours for the translator to finish its work--and going home disappointed. About a week later, I received a call that they'd finally figured out the bugs and I could pick up my translated program. Said program was more than half-again as large as the original in terms of object bytes. I was a bit skeptical of the Intel claim back then that the 8086 code was far more compact than its 8080 counterpart. And the blasted thing didn't work anyway when it was put to the test. […] There were other 80-to-86 translators for CP/M. I recall that Sorcim had one. AMC had a Z80-to-Z8000 translator as well. […] The [Intel] translator had several levels of translation […] there was the "literal", preserving detailed operation […] it was ISIS-II, running on an MDS-800, at, what, 2 MHz with 8" floppies. […]"
15. "Microsoft Macro Assembler (MASM) Unofficial Changelist" (http://bytepointer.com/masm/index.htm). *bytepointer.com*. 2018-08-21 [2016-09-08]. Archived (https://web.archive.org/web/20190717203656/http://bytepointer.com/masm/index.htm) from the original on 2019-07-17. Retrieved 2020-01-18. "[…] Tim Paterson […] had the following to say about his 8086 Assembler he wrote while at SCP: "The 8086 assembler I wrote originally was in Z80 assembly language and ran under CP/M. I believe it was distributed with the SCP 8086 CPU card as ASM86. I also wrote a translator that converted Z80 source code to inefficient but workable 8086 source code (Intel promoted this idea with a published translation table). This was called TRANS86 and was also written in Z80 assembly for CP/M. Once DOS was working, I applied the translator to ASM86 (and to itself) to create 8086 versions that ran under DOS. I do not have the change history in front of me […], but I believe that versions >= 2 marked the translated (DOS) version. If any history shows version numbers < 2, that was the CP/M version." […]"

16. Paterson, Tim (June 1983). "An Inside Look at MS-DOS - The design decisions behind the popular operating system - The history of and design decisions behind MS-DOS. how it works, and where it's going" (https://archive.org/details/byte-magazine-1983-06-rescan). *BYTE*. 16-bit Designs. Vol. 8 no. 6. McGraw-Hill, Inc. pp. 230–252. ISSN 0360-5280 (https://www.worldcat.org/issn/0360-5280). CODEN BYTEDJ (https://cassi.cas.org/searching.jsp?searchIn=codens&exactMatch=y&c=WIy460-R_DY&searchFor=BYTEDJ). Archived (https://web.archive.org/web/20170317042144/https://archive.org/details/byte-magazine-1983-06-rescan) from the original on 2017-03-17. Retrieved 2020-01-23. "[…] MS-DOS Design Criteria […] The primary design requirement of MS-DOS was CP/M-80 translation compatibility, meaning that, if an 8080 or Z80 program for CP/M were translated for the 8086 according to Intel's published rules, that program would execute properly under MS-DOS. Making CP/M-80 translation compatibility a requirement served to promote rapid development of 8086 software, which, naturally, Seattle Computer was interested in. There was partial success: those software developers who chose to translate their CP/M-80 programs found that they did indeed run under MS-DOS, often on the first try. Unfortunately, many of the software developers Seattle Computer talked to in the earlier days preferred to simply ignore MS-DOS. Until the IBM Personal Computer was announced, these developers felt that CP/M-86 would be the operating system of 8086/8088 computers. […]" [21] (https://archive.org/stream/byte-magazine-1983-06-rescan/1983_06_BYTE_08-06_16-Bit_Designs_djvu.txt)[22] (https://archive.org/details/byte-magazine-1983-06-rescan)
17. Seattle Computer Products (August 1980). "86-DOS - 8086 OPERATING SYSTEM - $95" (https://archive.org/stream/byte-magazine-1980-08/1980_08_BYTE_05-08_The_Forth_Language#page/n173/mode/2up). *BYTE* (Advertisement). Vol. 5 no. 8. BYTE Publications Inc. p. 173. ISSN 0360-5280 (https://www.worldcat.org/issn/0360-5280). CODEN BYTEDJ (https://cassi.cas.org/searching.jsp?searchIn=codens&exactMatch=y&c=WIy460-R_DY&searchFor=BYTEDJ). Archived (https://web.archive.org/web/20170405025551/https://archive.org/stream/byte-magazine-1980-08/1980_08_BYTE_05-08_The_Forth_Language) from the original on 2017-04-05. Retrieved 2013-08-18. "[…] 1. Read Z80 source code file written in CP/M format and convert to 86-DOS format. 2. Translator program translates Z80 source code to 8086 source code. 3. Resident assembler assembles the translated 8086 source code to 8086 object code. 4. Minor hand correction and optimization. (A recent 19K Z80 program translation took us about four hours to fix up. Even without optimization, it ran twice as fast as the original! […])" [23] (https://tech-insider.org/personal-computers/research/acrobat/8008-a.pdf)
18. Paterson, Tim (1994-10-03). "The Origins of DOS: DOS Creator Gives His View of Relationship Between CP/M, MS-DOS" (https://web.archive.org/web/20120531090452/http://www.ece.umd.edu/courses/enee759m.S2000/papers/paterson1994-kildall.pdf) (PDF). *Microprocessor Report*. MicroDesign Resources (MDR). **8** (13). ISSN 0899-9341 (https://www.worldcat.org/issn/0899-9341). Archived from the original (http://www.ece.umd.edu/courses/enee759m.S2000/papers/paterson1994-kildall.pdf) (PDF) on 2012-05-31. "[…] To get major software developers to port their products from the 8080/Z80 to the 8086, I decided we had to make it as easy as possible. I had already written a Z80-to-8086 source code translator (hosted on the 8080 and CP/M). My plan was that running an 8080 CP/M program through the translator would be the only work required by software developers to port the program to the 8086. In other words, the interface used by applications to request operating system services would be exactly the same as CP/M's after applying the translation rules. […]"

19. Paterson, Tim (2007-09-30). "Design of DOS" (https://web.archive.org/web/20130120075653/http://dosmandrivel.blogspot.com/2007/09/design-of-dos.html). *DosMan Drivel*. Archived from the original (http://dosmandrivel.blogspot.com/2007/09/design-of-dos.html) on 2013-01-20. Retrieved 2011-07-04. "[…] CP/M Translation Compatibility […] For DOS to succeed, it would need useful applications (like word processing) to be written for it. I was concerned that SCP might have trouble persuading authors of application software to put in the effort to create a DOS version of their programs. Few people had bought SCP's 16-bit computer, so the installed base was small. Without the applications, there wouldn't be many users, and without the users, there wouldn't be many applications. […] My hope was that by making it as easy as possible to port existing 8-bit applications to our 16-bit computer, we would get more rogrammers to take the plunge. And it seemed to me that CP/M translation compatibility was what would make the job as easy as possible. Intel had defined rules for translating 8-bit programs into 16-bit programs; CP/M translation compatibility means that when a program's request to CP/M went through the translation, it would become an equivalent request to DOS. […] So I made CP/M translation compatibility a fundamental design goal. This required me to create a very specific Application Program Interface that implemented the translation compatibility. I did not consider this the primary API – there was, in fact, another API more suited to the 16-bit world and that had more capabilities. Both APIs used CP/M-defined constructs (such as the "File Control Block"); the compatibility API had to, and I didn't see a reason to define something different for the primary API. […] I myself took advantage of translation compatibility. The development tools I had written, such as the assembler, were originally 8-bit programs that ran under CP/M (CDOS). I put them through the translator and came up with 16-bit programs that ran under DOS. These translated tools were included with DOS when shipped by SCP. But I don't think anyone else ever took advantage of this process. […]"
20. "Z80 To 8086 Translator". *Z80/8086 Cross Assembler Release 1* (https://amaus.net/static/S100/seattle%20computer%20products/software/SCP%20Z80%208086%20cross%20assembler.pdf) (PDF). Revision A (Preliminary ed.). Seattle, Washington, USA: Seattle Computer Products. pp. 20–21. Retrieved 2020-01-18. "[…] The Seattle Computer Products Z80 to 8086 Translator runs on the Z80 under CP/M. It accepts as input a Z80 source file written using Zilog/Mostek mnemonics and converts it to an 8086 source file in a format acceptable to our 8086 Cross Assembler. To translate a file, simply type TRANS86 <filename>.<ext>. Regardless of the original extension, the output file will be named <filename>.A86 and will appear on the same drive as the input file. A file named TRNTEST.Z80 is included to demonstrate the translator. The entire Z80 assembly language is not translated. […]" [24] (https://archive.org/details/bitsavers_seattleComsemblerPreliminary_611077)
21. *86-DOS - Disk Operating System for the 8086. User's manual* (http://www.patersontech.com/dos/Docs/86_Dos_usr_03.pdf) (PDF). Version 0.3 (Preliminary ed.). Seattle, Washington, USA: Seattle Computer Products. 1980. Archived (https://web.archive.org/web/20190714004434/http://www.patersontech.com/dos/docs/86_Dos_usr_03.pdf) (PDF) from the original on 2019-07-14. Retrieved 2020-02-01.
22. Paterson, Tim (2013-12-19) [1982-07-01]. "Microsoft DOS V1.1 and V2.0: Z80 to 8086 Translator version 2.21 /msdos/v11source/TRANS.ASM" (http://www.computerhistory.org/atchm/microsoft-research-license-agreement-msdos-v1-1-v2-0/). Computer History Museum, Microsoft. Archived (https://web.archive.org/web/20191112060745/https://computerhistory.org/blogs/microsoft-research-license-agreement-msdos-v1-1-v2-0/?key=microsoft-research-license-agreement-msdos-v1-1-v2-0) from the original on 2019-11-12. Retrieved 2014-03-25. (NB. While the publishers claim this would be MS-DOS 1.1 and 2.0, it actually is SCP MS-DOS 1.25 and TeleVideo PC DOS 2.11.)

23. *SCP 86-DOS - Single-User Disk Operating System for the 8086* (https://archive.org/stream/bitsavers_seattleCominary_1042322/SCP_86-DOS_Preliminary_djvu.txt) (Preliminary ed.). Seattle, Washington, USA: Seattle Computer Products. 1980. Retrieved 2020-01-18. "[…] The source code translator can translate most Z80 source code into 8086 source code acceptable to the assembler after minor manual correction. This provides a relatively quick and easy way to transport programs between the processors. […] TRANS file […] The Z80-to-8086 Source Code Translator, called by this command, is essentially identical to our version that runs on the Z80, which is described in the back of the Assembler manual. The only differences: 1. The Translator is called TRANS, not TRANS86, and it runs on the 8086 under 86-DOS, not on the Z80 under CP/M. 2. The extension of the output file is "ASM", not "A86". […]" [25] (https://archive.org/details/bitsavers_seattleCominary_1042322)
24. *Z80 To 8086 Translator* (https://amaus.net/static/S100/seattle%20computer%20products/software/SCP%20Z80%20to%208086%20translator.pdf) (PDF). Seattle Computer Products. pp. TRANS-1–TRANS-2. Retrieved 2020-01-19. (23 pages)
25. Hughes, David B. (November 1982). "CP/M-86 And MS-DOS: A Comparative Analysis" (https://books.google.com/books?id=vy3cBZkjbZgC&pg=PA189). *PC Magazine*. Operating Systems. Vol. 1 no. 7. Software Communications, Inc. pp. 181–182, 187–190 [189]. Archived (https://web.archive.org/web/20200210202327/https://books.google.de/books?id=vy3cBZkjbZgC&pg=PA189&dq=combined+CP/M-80+DOS+opcode+program&hl=de&sa=X&ved=2ahUKEwjO7N2Q6MfnAhVS-6QKHTPADzcQ6AEwAHoECAAQAQ#v=onepage&q=combined%20CP%2FM-80%20DOS%20opcode%20program&f=false) from the original on 2020-02-10. Retrieved 2020-02-10. "[…] An impressive and useful array of software development utilities is a standard feature of MS-DOS. A program that translates 8080 or Z80 code into 8086 source code, a linker, and a library runtime combine with a powerful assembler to give the programmer everything needed to take full advantage of the PC's 16-bit processor. The MS-DOS translation program allows the user to translate code developed under CP/M-80 or SB-80 […] 8-bit operating system […] to MS-DOS 1.2 or 2.0. Some modification beyond simple translation may be necessary to get the programs to run on 16-bit systems, so I suggest that this tool be used primarily by a technically trained user. […]"
26. Garetz, Mark (1980-12-22). "According to Garetz…" (https://books.google.com/books?id=nD4EAAAAMBAJ&pg=PT11). *InfoWorld* *- News For Microcomputer Users*. **2** (23). Popular Computing, Inc. p. 12. ISSN 0199-6649 (https://www.worldcat.org/issn/0199-6649). Archived (https://web.archive.org/web/20200201163107/https://books.google.com/books?id=nD4EAAAAMBAJ&pg=PT11&lpg=PT11) from the original on 2020-02-01. Retrieved 2020-01-18. "[…] Last week was the semi-annual California Computer Swap Meet. This event is organized by John Craig […] Sorcim […] was debuting […] new products at the show […] Their other product was TRANS-86. TRANS-86 will take any CP/M compatible 8080/8085/Z-80 source code file and translate it into 8086 code. You can then assemble the new file with ACT-86. […]"
27. Blumenfeld, Dan (1982-12-04). "Z80 to 8086 translator" (http://mdfs.net/Archive/info-cpm/1982/12/04/040100.htm). Newsgroup: fa.info-cpm (news:fa.info-cpm). Archived (https://web.archive.org/web/20200116071746/http://mdfs.net/Archive/info-cpm/1982/12/04/040100.htm) from the original on 2020-01-16. Retrieved 2020-01-15. [26] (https://groups.google.com/d/msg/fa.info-cpm/QVCzDa3SAL0/WhE09zX4GxkJ)

28. "CompuPro" (https://archive.org/details/bub_gb_w_OhaFDePS4C). *PC: The Independent Guide To IBM Computers* (Advertisement). Vol. 1 no. 1. Software Communications, Inc. February–March 1982. pp. 70–71. Premiere/Charter issue. Retrieved 2020-01-23. "[…] WHY? FLEXIBILITY. CompuPro's 85/88 CPU runs CP/M 80, 86, MP/M II and MP/M 86. We offer WORDSTAR dBASE II SUPERCALC a host of languages, 8080 to 8088 translators and more! COMPATIBILITY. Our systems can use CP/M 2.2 utilities to write programs for the IBM PC. You simply create 8088 source (either write it with your favorite CP/M 80 editor or translate it with Sorcim's TRANS 86), cross-assemble your source (with Sorcim's ACT86), link your hex file (with CP/M 80's LOAD command), translate it to the IBM PC (with G&G's CPM-IBM program), and run it on your IBM PC! This procedure DOES NOT require MS-DOS! […] WHY? FLEXIBILITY. CompuPro's 85/88 CPU runs CP/M 80 & 86 or MS-DOS. We offer WORDSTAR, dBASE II, TRANS86, XLT86, ACT86, SUPERCALC, CBASIC, MBASIC, MFORTRAN and more! […]" [27] (https://archive.org/stream/bub_gb_w_OhaFDePS4C/bub_gb_w_OhaFDePS4C_djvu.txt)[28] (https://archive.org/download/bub_gb_w_OhaFDePS4C/bub_gb_w_OhaFDePS4C.pdf)
29. Warren, Jr., Jim C. (July 1982). "Sorcim' Somethin' " (https://amaus.net/static//S100/MAGAZINE/silicon%20gulch/30%20198207%20silicon%20gulch.pdf) (PDF). *Silicon Gulch Gazette*. Rumors Mongered Here. Woodside, California, USA: Computer Faire. **7** (30): 1, 2, 4, 6, 11, 14, 15 [11]. Retrieved 2020-01-15. "[…] Sorcim just completed the purchase of ISA. […] They have also had an 8080-to-8086 translator - Trans-86 - operational for over a year […]"
30. Barry, Tim (1982-04-05). "XLT-86, a CP/M utility program by Digital Research" (https://books.google.com/books?id=ZjAEAAAAMBAJ&pg=PA40). *InfoWorld - The Newsweekly for Microcomputer Users*. Software Review. **4** (13). Popular Computing, Inc. pp. 40–41, 53. ISSN 0199-6649 (https://www.worldcat.org/issn/0199-6649). Archived (https://web.archive.org/web/20200201160327/https://books.google.com/books?id=ZjAEAAAAMBAJ&pg=PA40&lpg=PA41) from the original on 2020-02-01. Retrieved 2020-01-25. "[…] XLT-86 1.1 […] XLT-86 is an analytical translator program written in PL/I-80. It reads the entire 8080 source program, assembles it to machine code, analyzes the register, memory and flag utilization, and emits an optimized 8086 assembly-language program. […] There is also a version of XLT-86 for those of you who have access to a VAX 11/750 or 11/780. This version can translate much larger programs. It also costs $8000. […] While the translator adds some labels and equates to the source program as part of the translation, all original comments and program labels are passed through intact to the translated program. […] The program translation proceeds in a five-step process. First, the program is scanned and assembled to produce symbol values and locations. Second, the program structure is analyzed and decomposed into basic blocks. Third, the basic blocks are analyzed to determine program flow and resource usage. Forth, the block structure and register allocation data is gathered into a listing for the user. Fifth, the flow information and source program are used to produce the 8086 source program. […]"
31. Kildall, Gary Arlen (1982-04-19). Swaine, Michael; Freiberger, Paul; Markoff, John Gregory (eds.). "Digital Research founder discusses his view of the business" (https://books.google.com/books?id=YzAEAAAAMBAJ&pg=PA23). *InfoWorld - The Newsweekly for Microcomputer Users*. Special section: CP/M. **4** (15). Popular Computing, Inc. p. 23–24. ISSN 0199-6649 (https://www.worldcat.org/issn/0199-6649). Archived (https://web.archive.org/web/20200201160838/https://books.google.com/books?id=YzAEAAAAMBAJ&pg=PA23&lpg=PA23) from the original on 2020-02-01. Retrieved 2020-01-17. "[…] Kildall: […] A year and a half ago I was probably spending 75% of my time on the business and 25% on programming. XLT-86 was a product I was working on at that time, and it took me nine months to do it. That would have been a three-month project if I had been able to concentrate on it. […]"

32. Kildall, Gary Arlen (June–July 1982). Bunnell, David Hugh; Edlin, Jim (eds.). "Gary Kildall - The Man Who Created CP/M: CP/M's Creator - An Indepth PC-Exclusive Interview with Software Pioneer Gary Kildall" (https://archive.org/stream/PC-Mag-1982-06/PC-Mag-1982-06_djvu.txt). *PC Magazine*. Operating Systems. Vol. 1 no. 3. Software Communications, Inc. pp. 32–38, 40 [35]. Retrieved 2020-01-17. "[…] PC: What are some of the complexities involved in translating a program from 8080 to 8086 form? Kildall: Straight translations at the source program level you can do pretty much mechanically. For example, an 8080 "Add immediate 5" instruction turns into an "Add AL 5" on the 8086 — very straightforward translation of the op codes themselves. The complexity in mechanical translation comes from situations such as this: The 8080 instruction DAD H takes the HL register and adds DE to it. For the 8086 the equivalent instruction would be something like ADD DX BX, which is fine, no particular problem. You just say the DX register is the same as HL and BX the same as DE. The problem is that the 8086 instruction has a side effect of setting the zero flag, and the 8080 instruction does not. In mechanical translation you end up doing something like saving the flags, restoring the flags, doing some shifts and rotates, and so forth. These add about five or six extra instructions to get the same semantic effect. There are a lot of sequences in 8080 code that produce very strange sequences in 8086 code; they just don't map very well because of flag registers and things of that sort. The way we get software over is a thing called XLT-86. It's been out six months or so. PC: By "better" code do you mean smaller? Kildall: Twenty percent smaller than if you just took every op code and did a straight translation, saving the registers to preserve semantics. PC: How does the size of the translated program compare to the 8080 version? Kildall: If you take an 8080 program, move it over to 86 land and do an XLT-86 translation, you'll find that it is roughly 10 to 20 percent larger. With 16-bit machines it's more difficult to address everything; you get op codes that are a little bit bigger on the average. An interesting phenomenon is that one of the reasons you don't get a tremendous speed increase in the 16-bit world is because you're running more op codes over the data bus. […]"

33. Huitt, Robert; Eubanks, Gordon; Rolander, Thomas "Tom" Alan; Laws, David; Michel, Howard E.; Halla, Brian; Wharton, John Harrison; Berg, Brian; Su, Weilian; Kildall, Scott; Kampe, Bill (2014-04-25). Laws, David (ed.). "Legacy of Gary Kildall: The CP/M IEEE Milestone Dedication" (https://archive.computerhistory.org/resources/access/text/2014/06/102746909-05-01-acc.pdf) (PDF) (Video transscription). Pacific Grove, California, USA: Computer History Museum. CHM Reference number: X7170.2014. Archived (https://web.archive.org/web/20141227142045/http://archive.computerhistory.org/resources/access/text/2014/06/102746909-05-01-acc.pdf) (PDF) from the original on 2014-12-27. Retrieved 2020-01-19. "[…] Rolander: I mentioned earlier that Gary liked to approach a problem as an architect. […] And he would draw the most beautiful pictures of his data structures. […] And when he finished that […] and was convinced those data structures were now correct, he would go into just an unbelievable manic coding mode. He would just go for as many as 20 hours a day […] he was just gone during these periods of time. On a couple of those occasions, when he'd get something running the first time, which could be in the middle of night. And all you who have written software have seen that, for example, that the first time it comes up on the screen, you've got to tell somebody. My wife Lori will tell you that I had a couple of those calls in the middle of the night, LOGO was one example, XLT 86 was another, where he got it running the first time, and he had to have somebody see it. So it didn't matter what time it was, he'd call me, I'd have to come over and see it running. […]" [29] (https://web.archive.org/web/20191002192143/https://ethw.org/Milestones:The_CP/M_Microcomputer_Operating_System,_1974)[30] (https://www.youtube.com/watch?v=HO6IPpL0y8g) (33 pages)

34. "XLT86 Reduces Conversion Effort in Assembly Language Program Translation" (https://amaus.net/static/S100/software/DRI/Digital%20Research%20News/01x01%201981%20Digital%20Research%20News.pdf) (PDF). *Digital Research News - for Digital Research Users Everywhere*. Product Update. Pacific Grove, California, USA: Digital Research, Inc. **1** (1): 2, 7. November 1981. Fourth Quarter. Retrieved 2020-01-18. "[…] An 8- to 16-bit assembly code translator is now available from Digital Research. Called XLT86, it is designed to help ease the time-consuming process of converting CP/M software products from 8080- to 8086-based microcomputers. XLT86 can be used to translate any assembly language programs that are compatible with Digital Research's ASM, MAC or RMAC assembler format. The XLT86 program translator first reads an 8080 assembly language program and then produces an output file containing 8086 assembly language statements acceptable to the Digital Research ASM-86 assembler. Unlike other 8086 code converters that translate a single 8080 instruction into as many as ten 8086 instructions, XLT86 performs extensive data flow analysis to determine register usage throughout the original program. The information collected through this analysis is used during program translation to eliminate unnecessary flag save and restore operations. "The resulting 8086 program is both simpler and more compact than equivalent programs produced by other translators," according to Curt Geske, of the Digital Research marketing group. "Furthermore, XLT86 allows OEMs, end users and software vendors to preserve their investment in 8080-based assembly language programs when changing to 16-bit 8086-based computers by reducing the conversion effort." Programs translated by XLT86 run on both CP/M-86 and MP/M-86 […] XLT86 is available immediately. It operates on any 8-bit CP/M or MP/M system, or under the VMS operating-system for use on Digital Equipment Corporation VAX series mini-computers. The CP/M version is priced at $150. The VAX version sells for $8,000. […]"
35. Kildall, Gary Arlen (1973-10-01). "A Unified Approach to Global Program Optimization" (http://static.aminer.org/pdf/PDF/000/546/451/a_unified_approach_to_global_program_optimization.pdf) (PDF). *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. POPL '73. Boston, Massachusetts, USA: 194–206. doi:10.1145/512927.512945 (https://doi.org/10.1145%2F512927.512945). hdl:10945/42162 (https://hdl.handle.net/10945%2F42162). S2CID 10219496 (https://api.semanticscholar.org/CorpusID:10219496). Archived (https://web.archive.org/web/20170629213307/http://static.aminer.org/pdf/PDF/000/546/451/a_unified_approach_to_global_program_optimization.pdf) (PDF) from the original on 2017-06-29. Retrieved 2006-11-20. ([31] (http://portal.acm.org/citation.cfm?id=512945&coll=portal&dl=ACM))
36. Kildall, Gary Arlen (May 1972). *Global expression optimization during compilation* (Ph.D. dissertation). Seattle, Washington, USA: University of Washington, Computer Science Group. Thesis No. 20506, Technical Report No. 72-06-02.
37. *XLT86 - 8080 to 8086 Assembly Language Translator - User's Guide* (http://www.s100computers.com/Software%20Folder/Assembler%20Collection/Digital%20Research%20XLT86%20Manual.pdf) (PDF) (First printing ed.). Pacific Grove, California, USA: Digital Research, Inc. September 1981. Archived (https://web.archive.org/web/20161118230700/http://www.s100computers.com/Software%20Folder/Assembler%20Collection/Digital%20Research%20XLT86%20Manual.pdf) (PDF) from the original on 2016-11-18. Retrieved 2016-11-18. [32] (https://archive.org/stream/bitsavers_digitalResuideSep81_2620625/XLT86_Users_Guide_Sep81_djvu.txt)
38. "XLT86 for CP/M-80" (http://www.cpm.z80.de/download/xlt86.zip). Digital Research. Archived (https://web.archive.org/web/20200116055650/http://www.cpm.z80.de/download/xlt86.zip) from the original on 2020-01-16. Retrieved 2020-01-18. (NB. This ZIP archive contains the CP/M-80 executable XLT86.COM [22 KB] as well as two overlay files XLT00.OVL [8 KB] and XLT01.OVL [9 KB].)

39. Goldfarb, Ben (1982-12-09). "Re: 8080 to 8086 translation" (http://mdfs.net/Archive/info-cpm/1982/12/09/034933.htm). Newsgroup: fa.info-cpm (news:fa.info-cpm). Archived (https://web.archive.org/web/20200201161933/http://mdfs.net/Archive/info-cpm/1982/12/09/034933.htm) from the original on 2020-02-01. Retrieved 2020-01-18. "[…] The XLT86 program occupies approximately 30K bytes of main memory. The remainder of memory, up to the base of CP/M, stores the program graph that represents the 8086 program being translated […] A 64K CP/M system allows translation of 8080 programs of up to approximately 6K. […]" [33] (https://groups.google.com/d/msg/fa.info-cpm/f0lhZNqlBIs/A2P0PDLxrbMJ)[34] (http://www.retroarchive.org/cpm/cdrom/SIMTEL/ARCHIVES/CPM/8212-1.TXT)
40. Wharton, John Harrison (1994-08-01). "Gary Kildall, industry pioneer, dead at 52: created first microcomputer languages, disk operating systems" (http://tech-insider.org/personal-computers/research/1994/0801.html). *Microprocessor Report*. MicroDesign Resources Inc. (MDR). **8** (10). Archived (https://web.archive.org/web/20161118222925/http://tech-insider.org/personal-computers/research/1994/0801.html) from the original on 2016-11-18. Retrieved 2016-11-18. "[…] Ironically, many of the techniques Gary pioneered are being rediscovered now, ten years later. Apple and DEC are touting binary recompilation as a "new" technology for porting existing software to the PowerPC or Alpha architecture. Actually, DRI introduced an 8080-to-8086 binary recompiler in the early 1980s. […]"
41. "SPA Award to Dr. Gary A.Kildall: 1995 SPA Lifetime Achievement Award Winner" (https://www.digitalresearch.biz/kildallr.htm). Software Publishers Association (SPA). 1995-03-13. Archived (https://web.archive.org/web/20191221150356/https://www.digitalresearch.biz/kildallr.htm) from the original on 2019-12-21. Retrieved 2019-12-21 – via www.digitalresearch.biz. "[…] Kildall founded Digital Research, Inc. (DRI) in 1976, which is now part of Novell. […] In the 1980's, DRI introduced a binary recompiler. […]"
42. Swaine, Michael (1997-04-01). "Gary Kildall and Collegial Entrepreneurship" (https://web.archive.org/web/20070124184442/http://www.ddj.com/184410428). *Dr. Dobb's Journal*. Archived from the original (http://www.drdobbs.com/architecture-and-design/gary-kildall-and-collegial-entrepreneurs/184410428) on 2007-01-24. Retrieved 2006-11-20. "In March, 1995, the Software Publishers Association posthumously honored Gary for his contributions to the computer industry. They listed some of his accomplishments: […] In the 1980s, through DRI, he introduced a binary recompiler. […]"
43. 2500 A.D. Software (1984-12-11). "Super assemblers plus the world's largest selection of cross assemblers" (https://books.google.com/books?id=D0O_-ER3Z_gC&pg=PA166). *PC Magazine* (Advertisement). Vol. 3 no. 24. Englewood, Colorado, USA: PC Communications Corp. pp. 166–167. ISSN 0745-2500 (https://www.worldcat.org/issn/0745-2500). Archived (https://web.archive.org/web/20200201170955/https://books.google.com/books?id=D0O_-ER3Z_gC&pg=PA166&lpg=PA166) from the original on 2020-02-01. Retrieved 2020-01-24. "[…] 8086/88 Assembler With Translator $99.50 Available for MSDOS, PCDOS, or CP/M-86 […] This package also includes […] an 8080 to 8086 source code translator (no limit on program size to translate) […] Z-8000 Cross Development Package $199.50 […] This powerful package includes a Z-80/8080 to Z-8000 Assembly Language Source Code Translator […] The Translators provide Z-8000 source code from the Intel 8080 or Zilog Z-80 source code. The Z-8000 source code used by these packages are in the unique 2500AD syntax using Zilog mnemonics designed to make the transition from Z-80 code writing to Z-8000 easy […] 8086 and Z-8000 XASM includes source code translators […]" (NB. 8086/88 XASM available for Z-80 CP/M, Zilog System 8000 UNIX, Olivetti M-20 PCOS; Z-8000 XASM for Z-80 CP/M, MS-DOS, CP/M-86, Olivetti M-20 PCOS.)

44. "Zilog Unveils Modular Development System" (https://books.google.com/books?id=PKmCIHDeN6kC&pg=PA46). *Computerworld* - *The Newsweekly for the Computer Community*. **XIII** (34). Cupertino, California, USA: Computerworld, Inc. 1979-08-20. p. 46. ISSN 0010-4841 (https://www.worldcat.org/issn/0010-4841). Archived (https://web.archive.org/web/20200201165800/https://books.google.com/books?id=PKmCIHDeN6kC&pg=PA46&lpg=PA46) from the original on 2020-02-01. Retrieved 2020-01-24. "[…] Zilog, Inc. has introduced a series of modular and expandable product development systems (PDS) for Z8-, Z80-, and Z8000-based microcomputer designs. All four versions of the PDS 8000 system - models 10, 15, 25, and 30 - have […] A standard feature of each system is a Z8000 software development package, which includes the ZDOSII file management routine, PLZ/ASM high-level structured assemblers, a Z80/Z8000 translator and a Z8000 L and MACP macroprocessor. […] The models 10 and 25 have the same specifications as the models 15 and 30, respectively. but the 10 and 25 do not include the Z8000 development module. The PDS Model 10 is priced at $10,485; the Model 15 at $11,995; the Model 25 at $20,000; and the Model 35 at $21,500. All four systems are available 30 days after receipt of the order. […]"
45. Orlansky, Jesse, ed. (1979). *Proceedings 27–29 November 1979 - 1st Interservice/Industry Training Equipment Conference* (https://books.google.com/books?id=O7MgcxB_ZlYC&pg=PA413). The Center. p. 413. Technical report NAVTRAEQUIPCEN. Archived (https://web.archive.org/web/20200201170423/https://books.google.com/books?id=O7MgcxB_ZlYC&pg=PA413&lpg=PA413) from the original on 2020-02-01. Retrieved 2020-01-24. "[…] Table 1. 16-Bit Microprocessor Characteristics […] Zilog 8000 […] Software […] Zilog expects to support Z8000 with translators for PLZ, BASIC, COBOL and FORTRAN. These will permit conversion of Z80 code to Z8000 code, since Z8000 set is superset to Z80. […]"
46. *PDS 8000 Development System - The integrated approach to systems design* (https://archive.org/details/TNM_PDS_8000_development_system_for_systems_desig_20180331_0008) (Product Brief). Zilog. January 1980. Retrieved 2020-01-24. "[…] Z8000 TRANSLATOR: Provides a quick means to convert an existing Z80 Assembly Language program to Z8000 code and the PLZ/ASM program format. […]" [35] (https://archive.org/stream/TNM_PDS_8000_development_system_for_systems_desig_20180331_0008/TNM_PDS_8000_development_system_for_systems_desig_20180331_0008_djvu.txt)[36] (https://archive.org/details/TNM_PDS_8000_development_system_for_systems_desig_20180331_0008)
47. Thomas, Rebecca A.; Yates, Jean L. (1981-05-11). "Books, Boards and Software for The New 16-Bit Processors" (https://books.google.com/books?id=Cz4EAAAAMBAJ&pg=PA43). *InfoWorld* - *The Newspaper for the Microcomputing Community*. **3** (9). Popular Computing, Inc. p. 42–43. ISSN 0199-6649 (https://www.worldcat.org/issn/0199-6649). Archived (https://web.archive.org/web/20200201170121/https://books.google.com/books?id=Cz4EAAAAMBAJ&pg=PA43&lpg=PA43) from the original on 2020-02-01. Retrieved 2020-01-24. "[…] Digital Research has also announced plans for a Z8000 version of CP/M. Application software will be moved to the Z8000 when more development software is available. A commercially available translator from Z80 to Z8000 is needed. […]"
48. Guzis, Charles "Chuck" P. (2009-01-21) [2009-01-17]. "Re: CP/M or similar OS for 64K Z8002?" (http://www.vcfed.org/forum/archive/index.php/t-13845.html). *Vintage Computer Forum*. Genre: CP/M and MP/M. Archived (https://web.archive.org/web/20200116043046/http://www.vcfed.org/forum/archive/index.php/t-13845.html) from the original on 2020-01-16. Retrieved 2020-01-15. "[…] both Zilog and AMC offered Z80-to-Z8000 translation programs. Like the Intel 8080-to-8086 translator, it resulted in immediate bloat unless you were willing to hand-optimize the result. Much early MS-DOS code was auto-translated and tweaked 8080 CP/M code. I know that much of SuperCalc for the PC was, for example. Early (e.g. 3.3) versions of Wordstar for DOS probably also were. […] There were Z80-to-Z8000 source-code translators, but it wasn't a straightforward process ("strict" and "relaxed" modes; sometimes one Z80-to-several Z8000 instructions). The 8086 is much closer to the 8080 than the Z8000 is to the Z80. […]"

49. "2500 A.D. Software 8080/Z-80 to Z8000 source code translator v2.06b" (http://www.vintagecomputer.net/fjkraan/comp/mirror/z80cpu.eu/archive/rlee/L/LOOSECPM/300/U4/TRANS.COM). 2500 A.D. Software, Inc. 1982. Archived (https://web.archive.org/web/20200201170533/http://www.vintagecomputer.net/fjkraan/comp/mirror/z80cpu.eu/archive/rlee/L/LOOSECPM/300/U4/TRANS.COM) from the original on 2020-02-01. Retrieved 2020-01-24. [37] (https://web.archive.org/web/20151226171312/http://www.vintagecomputer.net/fjkraan/comp/mirror/z80cpu.eu/archive/rlee/L/LOOSECPM/300/U4/)
50. "802Z8000.ZIP 2500 A.D. Software 8080/Z-80 to Z8000 source code translator v2.06e" (http://pcsauro.altervista.org/CPMSW/lang/802Z8000.ZIP). 2500 A.D. Software, Inc. Archived (https://web.archive.org/web/20200201170814/http://pcsauro.altervista.org/CPMSW/lang/802Z8000.ZIP) from the original on 2020-02-01. Retrieved 2020-01-24. [38] (https://web.archive.org/web/20190912233429/http://pcsauro.altervista.org/CPM.PHP)
51. "List of languages that compile to JS" (https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS). Archived (https://web.archive.org/web/20200123055351/https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS) from the original on 2020-01-23. Retrieved 2018-03-11.
52. "JSweet" (http://www.jsweet.org/). Archived (https://web.archive.org/web/20191214170048/http://www.jsweet.org/) from the original on 2019-12-14. Retrieved 2019-12-14.
53. "Swiftify" (https://objectivec2swift.com/#/converter/code/).
54. "J2ObjC" (https://developers.google.com/j2objc/). Archived (https://web.archive.org/web/20191022074055/https://developers.google.com/j2objc) from the original on 2019-10-22. Retrieved 2019-10-22.
55. "Maia" (http://maia-eda.net). Retrieved 2020-05-13.
56. "C->Go translator" (https://github.com/rsc/c2go). Archived (https://web.archive.org/web/20181207133222/https://github.com/rsc/c2go) from the original on 2018-12-07. Retrieved 2018-01-11.
57. "Go 1.5 Release Notes" (https://golang.org/doc/go1.5). Archived (https://web.archive.org/web/20200201131555/https://golang.org/doc/go1.5) from the original on 2020-02-01. Retrieved 2018-01-11.
58. Cox, Russ. "Go 1.3+ Compiler Overhaul" (https://golang.org/s/go13compiler). Archived (https://web.archive.org/web/20200109183016/https://support.google.com/accounts/answer/32050) from the original on 2020-01-09. Retrieved 2018-01-11.
59. https://github.com/immunant/c2rust/blob/master/README.md
60. "Js_of_ocaml" (https://web.archive.org/web/20181208123603/http://ocsigen.org/js_of_ocaml/manual/overview). Archived from the original (http://ocsigen.org/js_of_ocaml/manual/overview) on 2018-12-08. Retrieved 2014-10-08.
61. *J2Eif Research Page - Chair of Software Engineering* (http://se.inf.ethz.ch/research/j2eif/). Se.inf.ethz.ch. doi:10.1007/978-3-642-21952-8_4 (https://doi.org/10.1007%2F978-3-642-21952-8_4). Archived (https://web.archive.org/web/20200101165402/http://se.inf.ethz.ch/research/j2eif/) from the original on 2020-01-01. Retrieved 2014-07-08.
62. "C2Eif Research Page - Chair of Software Engineering" (http://se.inf.ethz.ch/research/c2eif/). Se.inf.ethz.ch. Archived (https://web.archive.org/web/20200101165402/http://se.inf.ethz.ch/research/c2eif/) from the original on 2020-01-01. Retrieved 2014-07-08.
63. "Swiftify Objective-C to Swift Converter" (https://objectivec2swift.com/). Retrieved 2017-11-14.
64. "Runtime Converter" (http://www.runtimeconverter.com/). Archived (https://web.archive.org/web/20190710150000/http://www.runtimeconverter.com/) from the original on 2019-07-10. Retrieved 2017-11-14.

Further reading

- "Programming of Sycor Units Eased With TAL II" (https://books.google.com/books?id=tPYveN8W2qgC&pg=PA14). *Computerworld - The Newsweekly for the Computer Community*. **IX** (34).

Ann Arbor, Michigan, USA: Computerworld, Inc. 1975-08-20. p. 14. ISSN 0010-4841 (https://www.worldcat.org/issn/0010-4841). Archived (https://web.archive.org/web/20200201122051/https://books.google.de/books?id=tPYveN8W2qgC&pg=PA14&lpg=PA14&redir_esc=y#v=onepage&q&f=false) from the original on 2020-02-01. Retrieved 2020-01-18.

- Zerilli, Frank J.; Derouen, Craig (1986-12-11) [1985-08-15, 1984-12-20, 1984-11-20]. "8080 to 8086 ASM translator, with ASM source" (http://ftp.sunet.se/mirror/archive/ftp.sunet.se/pub/pc/mirror/simtelnet/msdos/asmutl/xlt86.zip). *SIMTEL*. XLT86.COM 1.10. Archived (https://web.archive.org/web/20200201123156/http://ftp.sunet.se/mirror/archive/ftp.sunet.se/pub/pc/mirror/simtelnet/msdos/asmutl/xlt86.zip) from the original on 2020-02-01. Retrieved 2020-01-18. 1984-11-11 version 1.05 (https://web.archive.org/web/20200201125052/http://msx2.org/NewPack/Mirrors/msx_ricardo_jurczky/imagem_msx_disco/F_/devel/asm/xlt86/) (NB. The DOS executable XLT86.COM [12 KB] translates Intel 8080 assembly language source code to Intel 8086 assembly language source code. Despite its name this implementation in 8086 assembly is *not* related to Digital Research's earlier and much more sophisticated XLT86.)
- Albo, Julián (2009-04-24). "Pasmo, ensamblador cruzado Z80 portable / portable Z80 cross assembler" (http://pasmo.speccy.org/) (in Spanish and English). Archived (https://web.archive.org/web/20200220234923/http://pasmo.speccy.org/) from the original on 2020-02-20. Retrieved 2020-02-20. "[…] Pasmo is a Z80 cross assembler […] that […] can generate object code in the following formats: raw binary, Intel HEX, PRL for CP/M Plus RSX, Plus3Dos (Spectrum +3 disk), TAP, TZX and CDT (Spectrum and Amstrad CPC emulators tape images), AmsDos (Amstrad CPC disk) and MSX (for use with BLOAD from disk in Basic). Starting with version 0.5.0 […] can also generate 8086 code from Z80 sources, in binary format for Ms-dos COM files or in CP/M 86 CMD format. […]" [39] (https://github.com/mkoloberdin/pasmo)[40] (https://github.com/mrcook/Z80Assembly/blob/master/docs/pasmo.md)
- Ciarcia, Steve (May 1984). "Trump Card - Part 1: Hardware - Speed up your IBM PC with 16-bit coprocessing power" (https://www.americanradiohistory.com/Archive-Byte/80s/Byte-1984-05.pdf) (PDF). *BYTE - The small systems journal*. Ciarcia's Circuit Cellar. Vol. 9 no. 5. McGraw-Hill, Inc. pp. 40–52, 54–55. ISSN 0360-5280 (https://www.worldcat.org/issn/0360-5280). Retrieved 2020-01-29. "[…] It instead executes programs written in high-level languages such as BASIC and C (a Pascal compiler and a 8088 to Z8000 translator are in the works. […]" and Ciarcia, Steve (June 1984). "Trump Card - Part 2: Software - TBASIC and C compilers and an assemble" (https://www.americanradiohistory.com/Archive-Byte/80s/Byte-1984-06.pdf) (PDF). *BYTE - The small systems journal*. Ciarcia's Circuit Cellar. Vol. 9 no. 6. McGraw-Hill, Inc. pp. 115–122. ISSN 0360-5280 (https://www.worldcat.org/issn/0360-5280). Retrieved 2020-01-29. "[…] I expect that object-code translators for Z80-to-Z8000 and 8088-to-Z8000 conversions will soon be available […]", also available as Ciarcia, Steve (1990). "Trump Card - Part 1: Hardware - Speed up your IBM PC with 16-bit coprocessing power & Part 2: Software - TBASIC and C compilers and an assemble" (https://books.google.com/books?id=fBuiNpYlyHcC&pg=RA2-PA138). In Gonneau, Daniel; Bernardi, Fred; Ausburn, Richard (eds.). *Ciarcia's Circuit Cellar*. **7**. McGraw-Hill Publishing Company. pp. 138–152, 153–160. ISBN 0-07-010969-9. Archived (https://web.archive.org/web/20200201125917/https://books.google.com/books?id=fBuiNpYlyHcC&pg=RA2-PA138&lpg=RA2-PA140) from the original on 2020-02-01. Retrieved 2020-01-29. [41] (https://amaus.net/static/S100/zilog/z8000/BYTE%20Trumpcard%20Z8000.pdf)
- Microcontroller Division Application Team (2000). "Translating Assembly Code From HC05 To ST7" (https://www.st.com/content/ccc/resource/technical/document/application_note/4f/89/a9/95/41/e7/4f/4c/CD00004128.pdf/files/CD00004128.pdf/jcr:content/translations/en.CD00004128.pdf) (PDF) (Application Note). STMicroelectronics. AN1106/0200. Retrieved 2020-01-18. (9 pages) (NB. This software translator was developed by ST and translates Motorola 6805/HC05 assembly source code in 2500AD Software format into ST7 source code. The MIGR2ST7.EXE executable for Windows is available from "MCU ON CD".)

External links

- Chaudry, Gabriele "Gaby" (2009-07-11). "Das Intel zu Zilog - Übersetzungsprojekt" (http://www.prof80.de/i2zkurz.html). *I2Z-Translator* (in German). Archived (https://web.archive.org/web/20160911054757/http://www.prof80.de/i2zkurz.html) from the original on 2016-09-11. Retrieved 2020-01-18.
- "PortAsm Assembler to Assembler Translation" (http://microapl.com/Porting/PortAsm.html). MicroAPL Ltd. 2017 [1996]. Archived (https://web.archive.org/web/20190730200919/http://microapl.com/Porting/PortAsm.html) from the original on 2019-07-30. Retrieved 2020-01-18.
- "Our Methodology - The Source to Source Conversion Process" (http://www.mpsinc.com/process.html). Micro-Processor Services, Inc. (MPS). Archived (https://web.archive.org/web/20190512171423/http://www.mpsinc.com/process.html) from the original on 2019-05-12. Retrieved 2020-02-01.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Source-to-source_compiler&oldid=1036892984"