The RASSIRIS project
Simulator of ASSIRIS jobs on Felix-C computers

Alexandru Dan Corlan, MD, PhD
alexandru@corlan.net

January 22, 2024
doi: 10.5281/zenodo.10245243

Contents

1 Introduction

The Felix-C series, the prototype of which was Felix-C 256, were mainframes produced in Romania that were clones of the french IRIS-50 computers, that were, at their turn, licensed as modified, not binary compatible, designs from SDS Sigma-7 and Sigma-9 computers. See section 1.5 for details.

Felix-C were designed in the late 1960s and early 1970s and produced in Romania until 1980. They were 32 bit computers, with paged memory management and 64 bit arithmetic double precision floating point in hardware. A somewhat comparable design in the microprocessor world, from a conceptual point of view, is the Motorola 68020 that appeared in 1984. Of course, the Felix/Iris/Sigma were using earlier, TTL low scale integration (chips with individual gates and bi-stables) on hundreds of boards in large racks, core magnetic memory and punched card readers.

Besides the C256, the prototype, 256Kb memory model, a reduced one, C32, was also built, which had only up to 64Kb of memory and was destined for process control. Larger ones, C512 and C1024 were later introduced, with a slightly extended instruction set and up to 1Mb of memory.

The clock was about 4MHz, but it was not a pipe-lined architecture and the execution of most instructions took around 3 microseconds, with some 64 bit arithmetical ones taking up to 55 microseconds [8].

Almost all traces of the Felices disappeared and it is not possible to find a non-working relic, much less a working one, and, as of November 2023, we could find no binary software. However, a number of books on the hardware, operating system (Siris-2 and Siris-3) with the SGF file system, ASSIRIS assembler with system call macros, the Socrates database, FORTRAN, COBOL, the Magiris microprocessor, can still be found.

1.1 The Felix culture

An important effect of the Felix computers was the technical culture that developed around this specific architecture, and ASSIRIS in particular. The Felices were almost the sole computers available in Romania for a decade or more, and the sole system that was taught in schools and universities. Even though relatively few people ran relatively few jobs on the really few computers that were actually functioning, this architecture was synonymous with computing for over a generation.

Six years after Felices stopped being produced, in 1986, while various mini and microcomputers were beginning to appear, the national computing Olympics for high-school pupils still required ASSIRIS as a skill (`https://vang.hq.ro/2023/11/27/it-in-romania-inainte-de-89/`).

1.2 Importance of the simulator project

The primary use of this simulator is as a museum piece. What it aims to preserve is the experience of running some ASSIRIS jobs on the Felix-C computers, currently not the structure of the computer itself. However, the structure of similar computers, the SDS Sigma-7 and Sigma-9 is preserved in SimH and that of relatively similar computers, the S/360 from IBM, is preserved in the Hercules simulator—and, of course, binary compatible IBM mainframes are still in broad use.

In other terms, what we aim to reproduce and preserve is not necessarily the Felix-C computer in detail, per se, but the substrate of the dominant Romanian technical culture of computer programming in the 1970s and 1980s, which was focused on developing and running user-space jobs on these computers, and obtaining printed outputs.

1.3 Aim of the project, what we try to preserve

Using and programming computers, 50 years ago, was different in some ways from what we do today. We are not referring only to the hardware (tapes, front panel, card readers and punchers, lack of interactive displays) but to the way software was conceived.

The assembler was not supposed to be only the target of compilers or an unusual language in which you encoded some small, performance critical, part of your program, or a driver, like today.

ASSIRIS was supposed to be used mostly by humans, and was the main method of controlling the computer, the equivalent of the shell (such as bash) in unices like Linux, today. There was a plethora of system macros that performed various common tasks, such as copying or printing files, or starting other programs, that were directly integrated in the assembler.

Programs had a segmented structure, because of the specifics of addressing. You had local address references, for calls, jumps and variables, that were inside segments up to 64K. There were also far calls and accesses, that went through a kernel trap and were invoked with macros. 64Kbytes meant a lot of memory.

This segmented structure made part of programming and thinking. High level languages, FORTRAN and COBOL, were used for very specialized purposes and also had to implement the segmented structure of the program and in-memory data. The segments actually provided an early modularisation and information hiding feature.

Segmentation was also found in the S/360 and probably many other systems of the time and maybe it can be traced to the organization of symbol visibility in C program files, although the 64K limit for a an object file image has long disappeared.

We are today used to flat and huge address spaces. However, as our computers and clusters have more and more processors of all kinds, it is possible that segmented programming reemerges somehow, under different names—as in the GPU "kernels".

It is this specific experience of the assembler-shell, especially in batch computing were one job consisted of all the program and all the data as one input and that deterministic-ally produced one output, that is rare today, at least in the personal computer world.

Another feature was that you could know your computer, hardware and software, system and standard applications with a relatively limited effort, perhaps a couple years of training and experience. We don't have this anymore today.

Also, the assembly language was very orthogonal and simple. This was probably, primarily, because decoding circuitry was expensive, but it helps clarity and predictability a lot, after you overcome the peculiarity of the syntax. The syntax is difficult because small numbers in different places mean quite different things than other numbers in the same places. Conditions for jumps, for example, come as numbers that usually denote registers but in this case do not mean registers–but conditions.

The mapping of the registers in the first 64 bytes of memory allows simple techniques that are not available anymore in modern processors, such as iterating through the registers. In modern interpreted bytecode languages, however, registers are actually in the memory, except access to them using memory addresses is not available despite the fact that normal access via instruction is as slow as memory access.

1.4 Why learn ASSIRIS and use the Felix simulator

Frankly, there are very few important reasons to try ASSIRIS, other than nostalgia or curiosity.

By learning ASSIRIS and running a few jobs you may:

1. experience the relatively unusual syntax and addressing modes of the Felix/Iris/Sigma processors, in which the registers were the first bytes of memory and there were various indirect and indexed

modes that also used the memory/register overlap–although you could do that with many SimH machines almost as well;

2. relive a time when the computer, its operating system, operating instructions, languages could be described in a book of 1000 pages that contained everything, that was all you could rely on that–although there are other ways to relive that with the ZX Spectrum, the Amiga or the Archimedes, not to speak about the Oberon;

3. experience the design of computing job-files where all the data and programs were combined together; this is somewhat enlightening with respect to reproducibility problems we encounter even today–although there are other job-file languages and mechanisms that are more widely available and better documented and tested, such as S/370 systems on Hercules emulators;

4. if you were learning computer science in Romania (and possibly in other countries) in the 1970s and 1980s but, frustratingly, did not have the opportunity to actually run programs on Felices, this can really help with finally healing that frustration (except only ASSIRIS-wise; COBOL-68 and FORTRAN-IV fans must wait a little more);

5. compare running jobs on the Felices and on z80/8080/8086/68k or on other computers and assess whether the decision to not continue the Iris-80/Felix/Sigma lines of computers, by integrating them into microprocessors–thus allowing the continuation of the technical culture that developed around them–was a good, or a bad decision;

6. procrastinate—when you don't know how to avoid a potentially scary task, it may be better to develop a couple of tests for a useless simulator, such as rassiris, rather than, for example, browse the internet or solve crossword puzzles;

7. have fun (sort of).

1.5 History of ASSIRIS and Felix-C computers

The Felix-C-256 computer was a mainframe computer produced in Romania in the 1970in's (https://ro.wikipedia.org/wiki/Felix_C). It was more or less a copy of the IRIS-50/IRIS-80 computers produced in France by CII. It was produced by the 'Fabrica de Calculatoare' near Bucharest. Both the computer and the production technology were licenced from CII. It ran the SIRIS-2 operating system that ran also on the IRIS.

At their turn the IRISes were produced under licence from SDS (Scientific Data Systems, later XDS after acquisition by Xerox) being similar to the SDS Sigma-7 and Sigma-9 computers. There is a SIMH simulator of the Sigma computers, with an operating system (CP-V) in binary form that works like the original.

However, the IRISes and thus the Felixes, are not binary compatible with the sigmas, there are substantial difference detailed in subsection 3.1. Nevertheless, the ASSIRIS assembler, besides different mnemonics for instructions and other differences, bears a striking resemblance to the SDS/Xerox META-SYMBOL assembler, the reference manual of which is in reference [9]; however, it is not exactly the same and only about 3/4 of the directives were uptaken into ASSIRIS.

The Felix was the only computer available in Romania in the seventies and early eighties. Apparently, about 650 were produced[2]. Romania was a socialist country from the Warsaw treaty and there were no private enterprises. The Felices were to be found in state enterprises and institutions, including 'county computation centers' and could not be purchased privately. Their unreliability was legendary. However, they allowed the introduction of computer programming and education curricula relying on a specific, practical, existing and theoretically available computer system. One had something to actually talk about. A whole generation of computer engineers and programmers was introduced into the field on the Felix FORTRAN, COBOL and ASSIRIS, as well as Siris control language, starting from high school. Most of those who wrote programs on paper and discussed their functioning never had the opportunity to see a Felix in real life, much less to punch their jobs on cards and actually submit them for execution. For some, this created a lingering frustration.

While Felices remained in operation, or at least on inventories, until the end of the eighties, their place was overtaken in the Romanian enterprises by PDP clones, mostly with RSX and RT-11, then by 8080 and Z80 clones running CP/M or ZX spectrum operating systems or the probably locally developed SFDX, which was inspired by RSX, but for the Z80. Later, 8086 PC-compatible clones overtook the landscape. A hobbyist community also developed around home-made ZX Spectrum clones, CP/M and also CP/M-68k. Few UNIX installations were experimented here and there, until Linux appeared in 1992 (Romania joined back the free world in December 1989).

Fabrica de Calculatoare was caught by the 1989 revolution while trying to clone a VAX. It survived for another couple of years, than disappeared. The whole 'Pipera platform' (industrial campus) were it was found, together with other computing machinery factories and institutes was transformed and is today a technological campus for (mostly international) software companies.

Of the Felixes, almost nothing remained–except the people who had been initiated in computing and their students and a few programming manuals in antiquaries.

There are Corals and Independents still in operation in some informal museums, but no Felices. Almost nothing was to be found about the IRISes online either.

Recently, some tapes that seem to contain a few felix code files, source and possibly also binary, have been found and posted online (at `https://www.z80-romania.ro/Benzi_pentru_FELIX_CORAL_INDEPENDENT_etc_-170`).

2 About this project

The lack of binary system software, or any software, to test a simulator makes it impossible to produce a hardware simulator in the stile of SimH.

The purpose of this project is to produce a simulator of running assembly (ASSIRIS) jobs on a bare machine that is as similar as practically possible to the Felix-C-256. For this purpose, we do not reproduce the system portions of the Felix, but only the user-space.

2.1 History of this project

It started on November 3, 2023, on github (`https://github.com/dancorl/felix-ASSIRIS/commits/main/README.md`) as a hobby project. Some parts of this document appeared as the first readme version. The first usable version of the simulator, 0.1.0., was released November 11, 2023. Version 0.1.1. can be found also on zenodo: `https://zenodo.org/records/10117525`. The official page of the project is at `http://dan.corlan.net/software/emulators/felix-ASSIRIS/`.

The first version of this document was released at December 1, 2023, together with release v0.2.0 of the simulator.

A revised version of this document, with mostly typos and orthographic corrections, is posted on December 6, 2023.

The current version of the document is about v0.3.0 of the simulator, released january 21, 2024.

2.2 Road-map

Possible developments, and their names, are imagined below. There is a coding scheme, implementations starting with 'C', of which there should be only one (C256) with options, would represent attempts to reproduce the exact Felix machine. Implementations with 'CC' followed by the number of bits in the address field, are not necessarily exact implementations of C256, but approximations inferred from books. However, they try to remain as close as possible to the Felix. Implementations with 'CX' followed by the number of bits in the address field would be imaginary eXtensions of the Felix, in the style of MVS/Hercules/380 (https://mvs380.sourceforge.net/).

The only implementation being developed now is CC16, for the purpose of getting the instructions and addressing modes right.

C256 –would attempt to simulate exactly the Felix C256 with different memory configurations/availability; would aim for binary and even signal compatibility; possible only if we somehow recover the operating system and other binary files; these seem very unlikely at this time. One approach could be to transform a clone of the Sigma SimH simulator along the lines of the differences found in the literature. A non-compatible sigma architecture would result, that is closer somehow to the Felices, but for which no software is available and that is probably incompatible even with the Felix because we don't know all the differences; however, it would be a complete computer simulation.

CC16 is currently developed and described below; it aims to provide a minimal environment in which small ASSIRIS jobs can be run. It is a simplified, single 64K code segment job, with a minimal implementation of the HOST instruction, basically stdin, stdout; instructions are otherwise attempted implementations of the ones in the Felix, possibly with the exception of some system and character chain instructions; a minimum set of the directives and control cards are emulated, to make running stdin/stdout jobs in 64K possible. This is the fastest achievable, the simplest and probably sufficient for almost any demonstration purpose as well as for testing the instructions. Most examples that are found in existing manuals could be run in this system, once completed. This configuration is the first objective of this project and would amount to some preservation of the historical experience of running Felix/ASSIRIS jobs, while we learn how the system works. The limitation to 64K is not restrictive, as most Felix installation actually had 64K of memory or less—core memory was rare and very expensive—of which some was used by the operating system. The upgrade of this simulator to another configuration may involve some redesign of the simulator.

CC24 would attempt the implementation of the segmented architecture of SIRIS-3 jobs, including some monitor functions–such as CALLs into other segments, possibly some traps and user interrupts. Except for the HOST instruction, all instructions would still be from the Felix-C-256 or Felix-C-1024 set. The host instruction would be an extensive implementation, possibly with access to external files, at least sequential, and at least a librarian of jobs and sources and other features of the SIRIS operating system, for example some SGF calls. The maximum memory that can be handled is available. Multiprocessing, through time-sharing, as on the original processor could be available. References [3, 7] provide a sufficiently detailed description of the internal aspects of the operating system SIRIS to allow its approximate reproduction in the simulator. The CC24 machine would contain an as close approximation of the operating system as possible, however the OS would not be written in Felix machine code, but run in the simulator. The compilers for (segmented) FORTRAN and COBOL and the accepted language, including the peculiarities of interacting with the operating system are also described in other books, in sufficient detail to allow their re-implementation. This is a feasible, although larger, project with the available documentation.

CX32 A possible implementation of an imaginary upgrade of the Felix. New instructions and instruction formats would be defined, implementing a flat 4G memory model; they would include at least 32 bit immediate load/store and 32 bit jump. The segmented approach is optional, as the job can control 4Gb of RAM in one image. Could include: true multi-core multiprogramming with inter-process communication primitives; extensive implementation of files and libraries; networking with internet access, for example cards to declare web servers and to access files and other resources across the web (via the Ada Web Server that would be bound with the simulator); primitives to access multiple file formats and databases, including modern ones such as sqlite or hdf. Such a system would not be useful even with as a historical preservation, but could offer an insight into a contrafactual history narrative in which the Sigma/Felix line would have continued until today.

3 Known differences between systems

This section compares the RASSIRIS emulator with other emulators, with the Felix (as far as known) and with other computers.

The RASSIRIS is developed on an Intel/AMD64 processor, in Linux. It has not been tested on other processors or systems, although there is no obvious reason why it would not work.

3.1 Differences between Felix and Sigma computers

The Felix has a single, 32 bit instruction format.

The single instruction format has mostly the same fields, but not in the same order. In the Sigmas the order is: I,F,B,Q,D. In the IRISes and Felix it is I,B,Q,X,F,D. The Q field is named 'X' in the sigma documentation (`http://www.bitsavers.org/pdf/sds/sigma/sigma9/901733C-1_Sigma9_RefMan_Apr74.pdf`), but here we use Felix terminology. The Felix X field is a single bit that denotes the "indexed" addressing mode. In this mode–which, combined with the I (indirect) mode makes for addressing mode combinations–the Q register is added to the address formed by the contents of the B register and the D displacement, or to the address found in the memory word pointed to by (B)+D. The superposition between B and Q registers is also possibly different.

The opcodes F are also different, although the instructions are otherwise similar. Generally the opcodes perform the same functions, but the usual mnemonics are different as are the codes that correspond to the same actual instruction.

Otherwise, the addressing modes, the organization of the memory and memory management seem to be the same, and not very different from the IBM/S360. Larger programs are segmented, with a 'short' reference through the D field of the instruction and a longer reference using the (B)+D addressing scheme.

The D field is 16 bits in the Felices, thus segments have maximum 64K, and 17 bits in the Sigmas (because they don't have the indexed addressing flag), apparently with some schemes to extend that to up to 20 bits using alignment assumptions. Some, different, such assumptions may operate also in the Felices although they are still obscure to me. The IA (instruction address) field in the PSW in the Felix is 18 bits (thus the 256 in the Felix-C256, as 256K can be accessed with 18 bits), but the last two bits are always 0, as instructions are word-aligned. This schema of course would not work for data fields of instructions that deal with bytes or half words.

Otherwise, in the segmented view of the programs, the B (base) registers are usually used for the base of a segment, D for actual addresses in the segment and, if X is set, in the Felices (but not the sigmas), for an offset in a table inside the segment, a table that is pointed to by D.

The indirect addressing mode in the Felices also have a feature that I find bizarre, but probably do not understand exactly. It means that the computed address (say (B)+D) is replaced by the value found at that address. However, if that value is also 'indirect', that is it has the first bit set, another indirection is computed, taking that value into account and leading to a new indirection. This step is repeated up to 5 (five, exactly) times and then stops—there is specific circuitry in the computer that performs this stop. What is not clear is what happens if the final intended value is in fact a negative number that has bit 31 set. Very difficult bugs would seem to occur this way. I could not find such behavior in the Sigmas and the first version of the simulator performs exactly one indirection, as modern processors.

The dot, '.' is used on control cards in SIRIS rather than the exclamation mark, '!', that is used in CP-V.

3.2 Differences between current RASSIRIS language and the Felix-C-256

We list below the differences between CC and CX simulated machines on one side and the Felix-C-256 and the corresponding C256 simulated machine, if it will ever exist, on the other side.

1. Some system operations, such as channel I/O are not implemented; see chapters of individual versions for what is implemented and what not.

2. The float is IEEE instead of the Felix-specific one; all operations are normalized (as performed by the Intel/AMD64 processors).

3. in CC16, only a single segment, of 64K, with 'absolute' addressing, that is having B0 (which is always 0) as the default base register, is implemented.

4. Indirect addressing on the Felix would perform up to five indirections based on the value found at the target of each previous indirection; we do not implement this in the CC and CX, there is always one exact indirection.

3.3 Comparison of RASSIRIS simulators with SimH and QEMU

The SimH is an outstanding project of system preservation, as is QEMU—which in general deals with other, more modern processors. However, in QEMU, unlike in SIMH, one can run binaries on different processors (with different instruction set architectures) but system calls are to the Linux ABI, not to the old operating systems.

It you want to experience the old operating systems, you have run a machine, perhaps access it through some terminal and run commands that are executed by a copy of the operating system, and develop software using the tools that were available in the simulated system, exactly as they were (supposing you still have the binaries of such tools).

RASSIRIS is different, and somewhere in the middle. You develop programs with tools on a modern machine, for example Emacs in Linux. The assembler runs on the modern machine, and the objects that one used to interact with the Felix, such as tapes, printouts or card decks, are files on the modern machine. However, from inside the program you develop, you see everything as on the old machine, including some operating system calls, although more modern ones (semi-hosting API) are also available and the old are in fact mapped into the new. The operating system is simulated as is the assembler and the processor.

This approach was taken because we have documentation of the old system and processors but the binaries of the operating system, the assembler etc. It is also more comfortable than SIMH—but, of course, fails to preserve some dimensions.

4 The input language accepted by RASIRRIS

This section introduces the ASSIRIS language version accepted by the simulator, including instructions, macros, directives and control cards, as well as the machine code that is generated and the effects of its execution.

The simulator works as a Linux command that receives an input file–the format of which is specified here–and produces and output file. CC16 V1 will only read standard input and produce output. Other, more advanced versions, may deal with other files.

The input consists of lines, that for historical reasons are also called cards.

Unlike in the Felix, which used EBCDIC, the input and output lines are encoded in ASCII-8.

There is a scanned booklet with most instructions, directives and control cards of the original Felix at [6], that was used as inspiration for the ones below.

4.1 Types of input lines (cards)

The first character in each line determines what type of line it is.

The length of a line can be at most 80 characters, and ends with an ASCII 'LF' (10) character. The end of line character is not considered to be part of the line. The input line is treated as if it is padded with spaces until the 80th column.

Continuation lines are not currently supported.

Lines before the first .JOB card, between the .EOJ card and before the next JOB card, or after the last .EOJ card are ignored—considered comments.

The following types of lines, by character, may be in the input file, inside JOBs (as above), by the character they start with:

'.' Control cards, that start with a '.' are general instructions to the simulator–they match general instructions to the operating system in the Felices, such as JOB, EOJ, COMPILE, LINK, RUN;

'*' Comments start with '*' and are ignored.

'#,%' these lines are ignored like comment lines; '%' lines will introduce librarian control cards in the future.

A-Z a label of an ASSIRIS instruction starts this line

' ' a space may introduce an empty line, which is ignored, or an ASSIRIS instruction, macro or directive without a label;

Instructions are lines that directly translate into machine instructions, one to one, in the assembler output.

Macros are expressions that are translated into one or more machine code instructions.

Directives are lines directed to the assembler, that may generate binary output or not, or have other effects, but that do not correspond to machine instructions–such as declaring labels (EQU) or storing data (DATA).

Table 1: Overall (approximate) BNF syntax of the job-file.

```
<jobfile> ::= { [<comment>] <job> }* [<comment>]
<comment> ::= { <space-line> | <comment-line> }*
<comment-line> ::= { <comment-char> <char> * <end-of-card> }
<alpha-char> ::= 'A' .. 'Z' | 'a' .. 'z'
<num-char>   ::= '0' .. '9'
<hex-digit>  ::= '0' .. '9' | 'A' .. 'F'
<octal-digit> ::= '0' .. '7'
<binary-digit> ::= '0' | '1'
<alphanum-char> ::= <alpha-char> | <num-char>
<ident-char> ::= '%' | <alphanum-char>
<natural>    ::= <num-char>*
<identifier> ::= <alpha-char> <ident-char>*
<comment-char> ::= '*' | '#' | '(' | ')' | '[' | ']' | ';'
<ctrl-char> ::= '.'
<sp>     ::= { ' ' | '\t' } +
<end-of-card> ::= '\n'
<job> ::= <job-card> [ <conf-card>* ] [ <compile-block>* ]
                     [ <link-card> ] [ <run-block>* ] <eoj-card>
<job-card> ::= <ctrl-char> [<sp>] JOB <jobname>{',' <jobarg>} *
<jobname> ::= <identifier>
<jobarg> ::= 'AN:' <identifier> | -- account name
             'PN:' <identifier>   -- programmer/author name
<conf-card> ::= <ctrl-char> [<sp>] CONF <confarg>{',' <confarg>} *
<confarg> ::= 'MACHINE:' <identifier> |          -- machine, default: FELIX%CC16
              'VERB:' <natural>       |          -- verbosity level, default: 0
              'SPEED:' { O[RIGINAL] | N[ATIVE] } -- Original=Felix-c-256 timing,
                                                 -- Native=host speed
<compile-block> ::= <compile-card> <ASSIRIS-text> <end-card>  -- see next table
<eoj-card>      ::= <ctrl-char> [<sp>] EOJ
```

4.2 General structure of the input file

The input file is divided in jobs. Each job starts with a .JOB card and ends with an .EOJ card.

Inside a .JOB, there can be one or more sections of 'ASSIRIS text' that start with a .COMPILE card and end with an END card (which is a directive, not a control card). Typically, the END card is followed by .LINK and .RUN cards.

4.3 Machine code instructions

Machine code instructions are always 32 bits in length. They all have the same fixed format, described in table 4.

In the Felix literature, the R field is also called Q or X.

The Felix has, in fact, 17 registers. The 7 B registers, B1–B7 are the same with the last 7 general registers, R9–R15 (B+8). However, the B0 register is not the general R8 register, but a special register that is always 0. Thus, when you load a value in the R11 register, it is loaded also in the B3 register, it is the same register.

Table 2: Overall (approximate) BNF syntax of the job-file, the ASSIRIS text block

```
<compile-card>  ::= <ctrl-char> [<sp>] COMPILE ASSIRIS
<end-card>      ::= <sp> END [<identifier>]
<ASSIRIS-text>  ::= <ASSIRIS-expression> *
<ASSIRIS-expression ::= <comment-line> | <instruction> | <directive>
<instruction> ::=  [<label>]<sp><mnemonic>[,<reg>][<sp><insarg>][<sp><comment>]
<label> ::= <identifiers>
<mnemonic> ::= AD4i | AD4 | ...
<reg> ::= <natural> -- range 0..15
<insarg> ::= [<indirect>][<base> '.'][<argexpr>][','<index>]
<indirect> ::= '*'
<base> ::= 'B'<octal-digit>
<argexpr> ::= <label> | <half-constant>
<half-constant> ::= <decimal-hc>|<octal-hc>|<binary-hc>|<hex-hc>|<char-hc>
<decimal-hc> ::= ['+'|'-'] <natural>+
<octal-hc> ::= [''' 'O' ''']<octal-digit>+
<binary-hc> ::= [''' 'B' ''']<binary-digit>+
<hex-hc> ::=  [''' 'X' ''']<binary-digit>+
<string> ::= ''' <char> * '''
<char-hc> ::=  [''' 'C' ''']<char>[<char>]
<index> ::= <natural> -- range 0..15
<directive> ::= <org-dir>|<text-dir>|<textc-dir>|<data-dir>|
                <res-dir>|<csect-dir>|<align-dir>|<dsect-dir>|
<org-dir> ::= [<label>]<space> ORG <space><half-constant>
<csect-dir> ::= [<label>]<space> CSECT <space>['P'|'C'|'D']
<dsect-dir> ::= [<label>]<space> DSECT
<asect-dir> ::= [<label>]<space> ASECT
<system-dir>::= <space>SYSTEM<space><sys-symbol>
<sys-symbol>::= FELIX%CC16
<text-dir>  ::= [<label>]<space> TEXT <space> <string>
<textc-dir> ::= [<label>]<space> TEXTC <space> <string>
<data-dir>  ::=  [<label>]<space> DATA[,<size>[,<align>]] <constant>{,<constant>}*
<constant>  ::= <half-constant> | 'FS'<float-c> | 'FD'<float-c>
<float-c>   ::= ['+'|'-']<digit>* '.' <digit>* 'E' ['+'|'-']<digit>*
<size>      ::= <natural>
<align>     ::= <natural>
<align-dir> ::= [<label>]<space> BOUND <space> <natural>
<res-dir>   ::= [<label>]<space> RES[,<align>] <space> <natural>
```

Table 3: Overall (approximate) BNF syntax of the job-file, the link/run/dataset block

```
<link-card>     ::= <ctrl-char> [<sp>] LINK
<run-block>     ::= <run-card> [ <print-card> | <test-card> ] *
                    [ <dataset-card> <data-card>* <end-ds-card> ]
                    [ <print-card> | <test-card> ] *  -- not yet defined
<run-card> ::= '.' [<space>] RUN [<runarg>{',' <runarg>}*]
<runarg> ::= <ins-arg>
<ins-arg> ::= { KINS | MINS | GINS } ':' <natural>
<print-card> ::= '.' [<space>] PRINT <space> <p-option>{','<p-option>}*
<p-option> ::= 'HELP' | 'ABOUT' | 'SYMS' | 'LINKS' | 'DUMP' | MSG:' ... '
<dataset-card>  ::= <ctrl-char> [<sp>] DATASET
<data-card>     ::= <char>*
<end-ds-card>   ::= <ctrl-char> [<sp>] ENDDS
```

Table 4: The machine code instruction format

Field	From bit	To bit	Nr. bits.	Meaning
I	0	0	1	Indirect addressing
B	1	3	3	Base register, B0..B7
R	4	7	4	General register, R0..R15
X	8	8	1	Indexed addressing
F	9	15	7	Operation code (function)
D	16	31	16	Displacement

Table 5: Address calculation in the Felix-CC16 machine

I	X	Syntax	Address	Comment
0	0	MNE,R B.D	reg(B+8) + D	
		MNE,R D	D	B assumed 0, reg(8) = B0 = 0
1	0	MNE,R *B.D	(reg(B+8) + D)	object pointed to by addressed object
1	0	MNE,R *D	(D)	the absolute address contains a pointer
0	1	MNE[,0] B.D,R	reg(B+8)+D+reg(R)	R0 will be the other operand
0	1	MNE[,0] D,R	D+reg(R)	R0 will be the other operand
1	1	MNE[,0] *B.D,R	(reg(B+8)+D)+reg(R)	R0 will be the other operand
1	1	MNE[,0] *D,R	(D)+reg(R)	R0 will be the other operand

The same R registers are also used for single precision floating point operations.

Pairs of R registers, where the first register is even, that is R0,R1, or R10,R11, can be used as 64-bit registers, either for integer operations or double precision floating point operations.

The memory available to the job (user-space memory) is at addresses starting from 0 to 64K for CC16 and up to/ higher addresses in other configurations. Addresses are at the byte level, but instructions must be aligned at 4 bytes.

The first 64 addresses are, in fact, the registers. Thus, when the processor reads from these addresses, or writes to them, it reads/writes in the registers.

All instructions are register-to-memory (store) or memory-to-register (load, add etc). In order to perform register-to-register operations, you must use the first 64 addresses.

4.3.1 Address calculation

Most instructions consist of calculating an address, that we call A, and then performing an operation between two operands: a register R and the value found in memory at that address.

Some instructions, the ones with mnemonics ending in 'I', or immediate, do not perform on the value found at address A, but on the address A itself.

The address calculation is as explained in table 5. In indirect addressing (I=1), the Felix, if it finds a bit pattern that also represents indirect addressing (in a way that is still mysterious to this author, but it may involve the most significant bit being set), it will indirect again to the new pointer, than again, and again, up to 5 (five) times. As we consider this to be very difficult to control, we only implement the first indirection, as in more recent computers. An option in the C256 machine, if ever implemented, should reproduce the original repeated indirections (assuming they were ever used in available machine code).

4.4 Examples of addressing modes

```
       ORG X'0504'
ALPHA  DATA,4,4 X'100',X'200',X'300',X'400'

       ORG X'0100'
OMEGA  DATA,4,4 X'0999',X'0AAA,X'0BBB'
```

```
       ...
L1     LD4I,3  ALPHA
L2     LD4I,3 *ALPHA
L3     LD4,3   ALPHA
L4     LD4,3  *ALPHA
L5     LD4I,2  8
L6     LD4     ALPHA,2
L7     LD4    *ALPHA,2
L8     LD4I   *ALPHA,2
L8S    ST4,0   12
L9     LD4,8   12
L10    LD4,8  *12
```

ORG is a directives that establishes that the following code-generating instructions will start at address hex(adecimal) 0504 (decimal 1284), which is called the CAA (current assembly address). In CC16, this is an absolute address.

ALPHA is a label at the CAA, that is hex 0504.

DATA,4,4 means that the argument constants will be stored in successive locations, each of 4 bytes, aligned at 4 bytes. As the CAA, 0504, is already aligned at 4 bytes, it starts at this address. At 0504 the hex value 100 will be stored, that is hex 00000100. The data will be stored LSB first, that is 00, 01, 00, 00. At 0508, 00000200 will be stored, at 050C: 00000300, and at 0510: 00000400. The same applied to OMEGA.

Instruction L1 loads the R3 register with the address ALPHA, that is hex 0504. Instruction L1 is equivalent to:

```
L1 LD4I,3 B0.ALPHA
```

The address is the sum of B0—which is always 0 and the value of symbol ALPHA which is 0504.

Because it is an immediate instruction (notice the 'I'), the address itself is loaded into the register.

Instruction L2 is indirect. The address ALPHA (meaning B0.ALPHA) is used as a pointer to determine the effective address used in the computation. At ALPHA, we find X'0100' thus the effective address is 0100 hex. As the load is immediate, this address is loaded into R3 (on four bytes).

Instruction L3 is direct, but not immediate. The effective address is ALPHA as in instruction L1. However, the value that is loaded is that found at the effective address, that is X'0100'. Thus, L3 is equivalent to L2.

Instruction L4 is indirect *and* not immediate. The effective address is computed by adding ALPHA to B0 (which is 0) resulting ALPHA, then, because of the indirect mode, taking the value that is found, in memory, at address ALPHA, that is X'0100'. This X'0100' is the effective address. The L4 instruction loads the value found at X'0100' into R3, that is the value X'0999'.

L5 immediately loads 8 into R2.

L6 is indexed, but not indirect, and not immediate, addressing. The effective address is computed as the contents of B0 (that is 0) + ALPHA + the contents of R2, that is 8 from L5. Thus the effective address is X'0504' + 8 = X'050C'. Thus, the value loaded in R0 is the value at address X'050C' that is X'0300'. Being an indexed instruction, it operates with R0.

L7 is like L6, but also indirect. The effective address is computed by taking the value that is stored at B0+ALPHA, that is at ALPHA, where the value found is X'0100'. Now, to this value we add the

Table 6: Identified CSV calls

e1	meaning	impl
01	EXUP, Execute eXchange unit program	CC24
02	WAIT, for an external event specified by R2	CC24
03	RST, ? reset a blocked I/O request	CC24
04	TERM, R2=exit code, finish program normally	CC16
05	ABORT, R2=exit code, finish program abnormally	CC16
06	LDSG, B=segment number, R2, R3=?	CC24
08	SXR, set exception return, install an exception handler?	C256
09	VXR, validate exception return, return from handler?	C256
0A	TYPE, R2 is a control block	C256
0C	ASSIGN, assign to an i/o device	C256
11	RELEASE, unassign	C256
14	TIME, R2, R3 = HHMMSSNN where NN is in hundredth of seconds	C256

value in register R2 (indexed addressing), giving X'0108' which is the effective address. The value at this address, which is X'0BBB' is loaded into R0.

L8 proceeds like L7, except that, being an 'immediate' instruction, it loads into R0 the effective address itself, that is X'0108', not the value found there.

L8S stores register 0 into the location at address 12. However, 12 is in the 'registers' region and means the address of register R3 into which the value from R0 (X'0108') is stored.

L9 loads into register 8 (which is NOT B0, it is non-base register), the value found at absolute address 12, that is R3. At that address, the value X'0108' is found, which was stored by L8S.

L10 proceeds like L9, but the effective address is not the address 12, of the register R3, but the value found there (due to the indirect addressing mode), that is X'0108'. The value loaded into R8 is thus that found at X'0108', that is X'0BBB'.

4.5 The CSV instruction

The CSV instruction was a non-existent instruction, 2E, that would initiate a trap (diversion) and would be interpreted as a system call. According to reference [4] (p.91), the binary format of the CSV instruction was: e1,X'2E',e2, where e1 occupies the first byte of the instruction (I,B,Q), e2 occupies the D field (the last two bytes) and 2E is the code of instruction (F) *and* the X flag which is 0.

A number of macros compile into CSV calls. We identified them mostly in reference [8]. They are listed in table 6. The calls usually involved registers R2 and sometimes R3 for arguments.

In the CC16 simulator, we introduce a HOST instruction that has the same opcode as CSV, but the X flag is set. It was not present in the original Felix. It supplants the role of various traps, mostly for I/O purposes. It has an op-code like the other instructions, a syntax that is vaguely similar, the B, X=1, I, Q and D fields, but it is translated and interpreted differently from the usual Felix instructions. The syntax is:

```
HOST[,q] hostop[,arg[,b]]
```

Where: q is a general register (0..15), b is a base register (0..7, or 8..15, meaning the same registers, 0/8 is always 0). op and arg are 8-bit quantities and are stored in the D field of the instruction, op being the most significant and arg being the less. The I and X field are currently ignored.

This is equivalent to a function call of the hostop function, with up to three arguments: arg, b and q. b and q could represent the register numbers or the values inside the registers, depending on the hostop. The result of the operation is returned in q.

There is an informal industry standard on such calls, named ‘semi-hosting’, to which we map our call frame. The proposed calls in table 7 are based on `https://interrupt.memfault.com/blog/arm-semihosting`, with some additions. At the start, three file are open: 0, standard in–connected to the card reader at the line following the .RUN control card, 1, standard out, connected to the simulator standard output considered to be the system line printer and 2, standard error, a console/secondary printer connected to the simulator standard error.

The notation ’s/e’ in table 7 means success/error. An alternative implementation could be to return this in a flag (condition code), to avoid using a full register just for a bit and to allow immediate branching on condition. Apparently Z=0 for error and 1 for success in some CSV functions such as time, we could use it here as well.

AS.. reprezents an assertion about the the value of a register or memory address (ASEQ) or about condition codes (ASCx). If the suitable verbosity level is encountered, then the fact that the asserted condition/equality is met, or not, is signaled with a message at the console. The internal condition of the machine, apart from the next instruction address, is not changed.

4.6 The BSG instruction

The BSG is the other trap instruction used by Siris. It is, like the CSV, using the first byte (I,B,Q) as an argument, ignoring the X flag which is 0, the F=X’2F’ is the opcode and two arguments are in the D field. The first byte argument is the segment, the second byte argument is entry point in the segment. The first byte is not used in fact, just a bit is set to signify that the ’branch’ is linked to register 8 (that is, it would work like a BAL, not a BRU). It would probably be implemented in CC24, along with segmentation.

The syntax of the BSG is: BSG[,0—1] seg-and-entry where the seg and the entry are combined as a single, presumably hex, value—that is probably found by the assembler among the entry labels.

There is ample space for extensions in the binary space of this instruction. The BAL could be extended to all Q registers by allowing numbers up to 15 for the first argument. Setting the X flag could allow a long JMP that reaches a 24bit address range. However, as in fact only 22 bits are needed to reach a 24bit address range in jumps, because the instructions are 4-aligned, the remaining two bits could be used to implement 4 frequent jump conditions or calls or something.

5 The Felix-CC16 job simulator

Because the addressing modes, especially the immediate direct ones, favor addressing the first 64Kb of directly addressable memory, the first version of the simulator will focus on a machine with 64Kb. This should allow many of the ASSIRIS code examples from the literature to run.

The assembler, system directives, control cards and macros that were usual for ASSIRIS application users are implemented outside the simulated machine, but in the same executable as the simulator. Thus, we simulate a card reader for a ’job-file’, a sequence of JOBs on the standard input and the printer output on the standard output.

Table 7: Proposed semi-hosting calls

name	hostop	arg	b	q	meaning	res	impl
(from semihosting specification, adapted to felix)							
OPEN	01	filenr	name	opt	open file	s/e	CC24
ISTTY	09	filenr			check if tty,		?
WRITE	05	filenr	buffer	len	write block to file	s/e	CC16
READ	06	filenr	buffer	len	read block	s/e	CC16
CLOSE	02	filenr			close file	s/e	CC24
FLEN	0C	filenr			length of file	len	CC24
SEEK	0A	filenr		pos	seek in file	s/e	CC24
TMPNAM	0D				n/i		?
REMOVE	0E				n/i		?
RENAME	0F				n/i		?
WRITEC	03	filenr		char	write char to file	s/e	CC16
WRITE0	04	filenr	str		0term char to file	s/e	CC16
READC	07	filenr			read char	ch	CC16
CLOCK	10				ada clock in seconds	sec	CC16
ELAPSED	30	x			10**x ns since start	nr	CC16
TICKFREQ	31						?
TIME	11						?
HOSTCALL	12				call system command?		CX32
(newly added for the felix/ASSIRIS simulator; possibly)							
EXIT	60	exitcod		err	exit with code	---	CC16
PRINTF	68	format		arg	dec/hex/char/float	s/e	CC16
PUTC	69	ch			put character on stdo		CC16
RDCARD	6A	filenr	addr		read next line as card	s/e	CC16
WTCARD	6A	filenr	addr		write line as card	s/e	CC16
ALLOC	70	r/w	addr	len	alloc ram in 4G space	s/e	CX32
MMAP	78	r/w	addr	name	alloc ram read file	len	CX32
SAVE	79		addr	len	save mmap file	s/e	CX32
SETLEN	7A		addr	len	set new len to mmap fi	s/e	CX32
(debug instructions, create data that is transfered to the monitor)							
DUMP	C0	filenr	addr	len	dump memory and regs	s/e	CC16
ASEQ	D0	F0		val	assert val eq to const		CC16
ASCx	D0	0x			assert condition flags		CC16
name	hostop	arg	b	q	meaning	res	impl

The purpose is to simulate the experience a user would typically have had with using the Felix computer in ASSIRIS, without using the actual system tools, that are no longer available.

5.1 Imaginary outline of the future v1.0 version

This is a sketch of ideas about how the future 1.0 version would be. The purpose of CC16V1 is the simulation of the user experience with ASSIRIS programming of the Felix, at the level of small example programs of the kind found in manuals, without segmentation.

5.1.1 Outline of features

1. most instructions would be implemented, as similar as possible with the original, possibly with the following exceptions:
   (a) character string and decimal instructions, some system instructions, in particular i/o instructions, may not be implemented;
   (b) the floating point will be IEEE-754 floating point rather than the native one (with one byte exponent for both short and long) but most programmers, most of the time, would not feel any difference;
2. everything will be in ASCII; EBCDIC is a feature of the operating system, not the processor (except possibly the PACK/UNPK instructions and EDIT) and as there is no EBCDIC programs or data in existence, the experience of the programmer is mostly the same;
3. the operating system interface will be a newly added, simulated, trap instruction into the simulator, possibly named HOST, that will use the modern operating system in which it runs (Linux, currently) to simulate some system functions; initially, the only system functions are printing and exiting from the program; further functions may be added, perhaps following the 'semi-hosting' API.
4. macros, directives and control cards similar to those corresponding to the Sirius/SGF commands would be added; they will compile (macros) into HOST instructions or act on the setup created by the HOST;
5. a minimal set of system resources, such as possibly a source library and sequential files, will be simulated as directories on the hosting operating system and will be translated by the simulator;
6. some control cards, directives and macros for configuration, signaling, debugging etc, specific to the different environment of the host system;
7. minimum HOST operations and macros to allocate larger memory segments in the 24bit address space of the Felix or, perhaps, in the 32 address space allowed by its base registers, with macros–such as loading 32 bits immediately with a single instruction–that make its use comfortable; an MMAP instruction in the HOST could help also; this feature could be reserved for a future configuration, perhaps named Felix-CX32 (from the eXtended nature of the machine and 32-bit addresses–although the only extension is the fact that the artificial limit to 24 bit addresses is removed);
8. documentation, in a future version of this document, of the exact syntax-es and semantics of the instructions, macros, directives and control cards, as used in this simulator–that are intended to be the same as in the real machines, mostly, but may differ and are how they are.

5.1.2 Default file setup

The card reader is standard input of the simulator process.

The system printer is standard output of the simulator process.

The console is the standard error of the simulator process.

There is no console/front pannel input. The ‘machine’ is stopped or started through signals sent to the simulator process in the Unix (Linux) environment in which it runs.

5.1.3 Errors

Errors have the following levels:

0–9 warnings, issues that will not affect the functioning of the process, but may be of interest to the user/programmer

10–19 recoverable error, the process (compilation, execution) can continue but the results may be affected and should be discarded

20–29 unrecoverable error, the process cannot continue and will stop

30–39 errors in the simulator that may not belong to the user program

5.1.4 Verbosity levels

Verbosity levels are established with the .CONF control card, option VERB:nn, or by using the VERB:nn option in the .JOB card.

0 the system does not output anything at all except the output of the program, achieved through HOST instructions; warnings (error levels 0–9) are supressed. Errors level 10 and above are shown on the console. All listings, including as required by directives, are supressed.

10–19 warnings are shown, together with all error; usual messages from the system, including timing, assembly listings (if enabled), headers, are shown. Debugging messages are disabled. Unmet assersions are listed, but met assertions are not.

20-29 Debugging messages for the assiris program are enabled, including met assertions.

30-39 All debugging messages, including those for the felix simulator, put there during development, are displayed.

In general, messages have a level and are displayed if this is equal or above the verbosity level.

6 Specific features of each release

6.1 V0.3.0, CC16 only

In this version, most instructions—other than decimal, string, decrementing branch, some system instructions and i/o—are implemented. Additions include shifts and pseudoinstructions. There are many corrections, including extensive condition flag setting. The verbosity system was mostly implemented, including specific flags for each type of message. However, these flags are not yet available individually to the user. A job banner was added at the top of job listings.

Timings of all instructions were introduced, from reference [8].

A supervisor dealing with the HOST/CSV instruction was added. The HOST is implemented as a CSV with the X flag set. Assertions, ASEQ and ASCxx were added as macros for the HOST instruction, for testing equality of a register to a constant. The constants are stored in a system "assertions" (Asr) table that is accessible to the supervisor, but not to the machine program.

The following is from the felix.help file:

```
CSV and HOST are the same instruction (opcode) with different flags; do not
    use them individually, in general, use macros/pseudoinstructions
ASEQ,r val will check the assertion that general register r has the value val; it generates
    one HOST instruction that uses information (val) stored in a monitor table.
    When the host instruction is executed, it checks the value and produces a
    line in the output, depending on verbosity; a statistic of passed and failed
    assertions is also produced at the end with appropriate verbosity flags.
ASCZ ASCNZ ASCC ASCNC ASCS ASCNS ASCD ASCND ASCT ASCF ASEQ will check the condition
    and produce the appropriate reports depending on the verbosity level.
```

A few new tests, mostly from reference [4], were added, using assertions to check for proper execution.

An INS option was added to the RUN card, representing the number of instructions to be executed before running is supressed. New options are available in the CONF and JOB cards:

```
. JOB name,opt{,opt}*
. CONF opt{,opt}*
       JOB starts a job with a job name (max 8 characters) and options. CONF may change or pr
       at an earlier or latter time. Options can be: AN=cccccccc account name, PN=cccccccc
       programmer name, VERB=nn verbosity level--0=no system message; 10=basic messaging, inc
       20=extensive messages, incl passed assertions; 40=debugging messages., SPEED=N/O or 1/
       (default, as fast as possible), O or 0=original timing, 3--30 microseconds per instruc
```

Also from 'felix.help':

```
It implements the following instructions:
     AD4 AD4I AD8 ADF8 ADF4 ADH2 BRU BCF BCT BAL CP1I CP1 CP2 CP4 CP8 CSV
     DC4 DV2 DVU2 DV4 DVF8 DVF4 EO2 EO4 EX2 EX4 IC2 IC4 LDC2 LDC4 LD1I LD2I
     LD1 LDH2 LDL2 LD4 LD8 LDM LD4I MG2 MG4 MP2 MPU2 MP4 MPF8 MPF4 NF4 SB4I
     SB4 SB8 SBH2 SH2 SH4 SH8 SBF8 SBF4 ST1 ST4 ST8 STH2 STM
to which we added a couple more (see below): PRINT HALT CSV HOST
The following pseudoinstructions are also implemented:
     BC BGEZ BGZ BLEZ BLZ BNC BNOV BNZ BOV BZ NOP SLL2 SLL4 SLL8 SRL2 SRL4
     SRL8 SLC2 SLC4 SLC8 SRC2 SRC4 SRC8 SRA2 SRA4 SRA8 SLA2 SLA4 SLA8 ASCZ
     ASCNZ ASCC ASCNC ASCS ASCNS ASCD ASCND ASCT ASCF ASEQ TERM ABORT
```

6.2 V0.2.0, CC16 only

Purpose of this version: finalize addressing modes, more instructions, documentation (this document).

New developments:

- PRINT completed with 'f' and 'g' for float and long float and 'x' for hexadecimal printing of the contents of a register;
- DATA, TEXT and TEXTC directives implemented;
- BOUND instead of ALIGN (ALIGN also accepted);
- '*' as the current assembly location;
- the addressing modes, and their assembly syntax, established as in table 5;
- 64 bit operations implemented;
- floating point, short and long, implemented;

6.2.1 readme/installation

```
FELIX an approximate emulator for Felix-C-256 assembly (ASSIRIS) jobs
Copyright (c) 2023 Alexandru Dan Corlan, MD, PhD

This is release V0.2.0, December 1, 2023
It is part of the RASSIRIS project, to simulate Felix/ASSIRIS-like computers
http://dan.corlan.net/software/emulators/felix-ASSIRIS/

Detailed documentation is at:
http://dan.corlan.net/software/emulators/felix-ASSIRIS/rASSIRIS.pdf
and is also supplied in the distribution.

This code is released under the GNU General Public Licence version 2

INSTALLATION.
On LINUX. Install gnat, the GNU Ada Translator. On debian: apt install gnat

Untar the distribution: tar xvzf felix-ASSIRIS-v0.2.0

Change to the directory: cd felix-ASSIRIS-v0.2.0

Say: make

The resulting binary, felix, can be run locally or copied to /usr/local/bin
for general availability on your system.

RUNNING.

The simulator is a linux executable, felix, that takes a jobfile on
stdin and prints on stdout. Try: ./felix <hello_world.ASSIRIS
for an example.
```

6.2.2 The help page

With the command (job):

```
echo ". SAY 'HELP'" | ./felix
```

the following will be obtained:

```
***************************************************
*** *** FELIX/ASSIRIS VIRTUAL PROCESSOR HELP *** ***
***************************************************

================
= GENERALITIES =
================

FELIX V0.2.0., a felix/siris-2/ASSIRIS emulator, part of rASSIRIS project.
see http://dan.corlan.net/software/emulators/felix-ASSIRIS/ for details.

This version only has 64K of memory and only knows some of the instructions:
AD4 AD4I AD8 ADF8 ADF4 BRU BCF BCT BAL CP1I CP1 CP2 CP4 CP8 DC4 DV2 DVU2 DV4
DV8 DVF8 DVF4 EO2 EO4 EX2 EX4 IC2 IC4 LDC2 LDC4 LD1I LD2I LD1 LDL2 LDH2 LD4
LD8 LDM LD4I MG2 MG4 MP2 MPU2 MP4 MP8 MPF8 MPF4 NF4 ST1 ST4 ST8 STH2 STM
SB4I SB4 SB8 SBF8 SBF4
to which we added a couple more (see below): PRINT HALT
All addressing modes are implemented in this version, direct-nonindexed, indirect-nonindexed,
    direct-indexed and indirect-indexed.
The indirect addressing mode works for exactly one indirection.

=================
= CONTROL CARDS =
=================

The following cards (LIST and CONF) can appear anywhere in the input stream:
. LIST opt{,opt}* where opt can be:
       'HELP'     -- include this help text in the listing
       'ABOUT'    -- introduction to the FELIX/ASSIRIS system
       'SYMS'     -- the symbols table
       'LINKS'    -- the linkings performed by the link editor up to that point
       'DUMP'     -- `VIDAGE MEMOIRE' at that point
        MSG:'x'    -- display the message at that point
. CONF opt{,opt}* -- currently is ignored.

The following cards can only appear in a specifc sequence.
The sequence is: JOB, COMPILE, LINK, RUN, EOJ; then, you may repeat.
COMPILE must be follwed by ASSIRIS (. COMPILE ASSIRIS)
RUN will admit one option, KINS:n where n is the number of thousands of
  instructions to run. Without it, the simulator only runs 500 instructions
  then stops (as was necessary in early tests).
Otherwise, you may add any options to the cards, but they are currently ignored.

==============
= DIRECTIVES =
==============

Directives are cards that can occur only between '. COMPILE ASSIRIS' and 'END'
The directives: ORG, EQU, DS, DB, ALIGN, work as expected. EQU defines a symbol.
ORG changes the address (that must be its argument) where the assembler generates code
DATA,x,y datum{,datum}* stores each datum in succesive locations of x bytes each and
  aligned at y; by default, x and y are 4. Each datum has the syntax mentioned at DB
  below, to which FS:'<float>' and FL:'<float>' are added and, of course, the
  size of the objects specified can be of the size x.
DS is followed by a '-delimited string, the ASCII characters of which it puts into memory
  in succesive byte locations.
```

```
TEXT is like DC.
TEXTC is like TEXT, but the string is preceded by a byte containing its length.
DB is followed by a sequence of byte-sized numbers (0..255),
  Comma separated, that ar put into memory in sequence.
  the number can be: [-]ddd, decimal numbers; X'xx' hexadecimal, C'c' characters.
ALIGN and BOUND (synonymous). Before assemblying machine code,
  the assembly address must be aligned to 4 bytes.
  After DSs and DBs, always use ALIGN 4, if code follows.
  ALIGN take an optional argument--at what pace to align, in bytes. The default is 4.
END X must include this X which is the address, usually a label, where RUN will start execution.
CSECT is ignored.
Label expressions are not implemented, you can't say 'BRU ADDR+8' or something

====================
= NEW INSTRUCTIONS =
====================

HALT will finish running
PRINT,r will print the LSB of register r as an ascii character on stdout
PRINT,r C'd' will print the register r as a decimal on stdout
PRINT,r C'xy' will print the two ASCII characters x and y on stdout
```

6.2.3 Tests

The test files from the V0.1.1 should still work, hello world (subsection 6.3.3) and bogomips (subsection 6.3.4).

The following tests were added:

6.2.4 addrtest02.asr

Check the one below with section 4.4.

```
. JOB ADDRTEST,PN:DACORLAN
. COMPILE ASSIRIS

       CSECT
       ORG X'0504'
ALPHA  DATA,4,4 X'100',X'200',X'300',X'400'

       ORG X'0100'
OMEGA  DATA,4,4 X'0999',X'0AAA',X'0BBB'

L1     LD4I,3  ALPHA
       PRINT,3 C'1.'
       PRINT,3 C'3='
       PRINT,3 C'x'
       PRINT,3 C'\n'
L2     LD4I,3 *ALPHA
       PRINT,3 C'2.'
       PRINT,3 C'3='
       PRINT,3 C'x'
       PRINT,3 C'\n'
L3     LD4,3   ALPHA
       PRINT,3 C'3.'
       PRINT,3 C'3='
       PRINT,3 C'x'
       PRINT,3 C'\n'
L4     LD4,3  *ALPHA
       PRINT,3 C'4.'
       PRINT,3 C'3='
       PRINT,3 C'x'
       PRINT,3 C'\n'
L5     LD4I,2  8
       PRINT,3 C'5.'
       PRINT,3 C'2='
       PRINT,2 C'x'
       PRINT,3 C'\n'
L6     LD4     ALPHA,2
       PRINT,3 C'6.'
       PRINT,3 C'0='
       PRINT,0 C'x'
       PRINT,3 C'\n'
L7     LD4    *ALPHA,2
       PRINT,3 C'7.'
       PRINT,3 C'0='
       PRINT,0 C'x'
       PRINT,3 C'\n'
L8     LD4I   *ALPHA,2
       PRINT,3 C'8.'
       PRINT,3 C'0='
       PRINT,0 C'x'
       PRINT,3 C'\n'
L8S    ST4,0   12
       PRINT,3 C'8S'
       PRINT,3 C'.3'
       PRINT,3 C'=='
       PRINT,3 C'x'
       PRINT,3 C'\n'
L9     LD4,8   12
       PRINT,3 C'9.'
       PRINT,3 C'8='
       PRINT,8 C'x'
       PRINT,3 C'\n'
L10    LD4,8  *12
       PRINT,3 C'10'
       PRINT,3 C'.8'
       PRINT,3 C'=='
       PRINT,8 C'x'

       HALT
       END L1
. LINK
. RUN
. EOJ
```

6.2.5 addrtest.asr

```
. JOB ADDRTEST,PN:DACORLAN
. COMPILE ASSIRIS

         CSECT
STR      DS 'Hello World'
         DB 10,0
         ALIGN
STARSTR  DB 0,0,0,0
         ALIGN
RUNOW  LD8,6 STR
         AD4I,6 3
         ST8,6 STR
         LD4I,5 STR
         ST4,5 STARSTR
*HELLOW  LD4I,9 STR
HELLOW   LD4,9 STARSTR
DLOOP  LD1,4 *36
         CP1I,4 0
         BCT,8 NOPRINT
         PRINT,4
NOPRINT  AD4I,9 1
         CP1I,4 0
         BCF,8 DLOOP
         PRINT,3 C'..'
         PRINT,3 C'GA'
         PRINT,3 C'TA'
         HALT
         END RUNOW
. LINK
. RUN
* LIST MSG:'salut','SYMS','LINKS','DUMP'
. EOJ
```

6.2.6 datatest01.asr

```
. JOB DATATEST,PN:DACORLAN
. COMPILE ASSIRIS
```

```
       CSECT
       ORG X'0504'
ALPHA  DATA,4,4 FS'22.4e1',FS'11.2e1'

L1     LD4I,2 4
       LD4 ALPHA,2
       ADF4,0 ALPHA
       PRINT,0 C'f'
       HALT
       END L1
. LINK
. RUN
. EOJ
```

6.2.7 hello02.asr

```
. JOB HELLOWRL
. COMPILE ASSIRIS

         CSECT
STR      DS 'Hello World'
         DB 10,0
         ALIGN
HELLOW   LD4I,9 STR
DLOOP  LD1,4 B1.0
 CP1I,4 0
 BCT,8 NOPRINT
         PRINT,4
NOPRINT  AD4I,9 1
 CP1I,4 0
         BCF,8 DLOOP
 PRINT,3 C'..'
         PRINT,3 C'GA'
         PRINT,3 C'TA'
 HALT
         END HELLOW
. LINK
. RUN
. LIST MSG:'salut','SYMS','LINKS','DUMP'
. EOJ
```

6.3 v0.1.1, CC16 only

In this subsection we describe a first working release (0.1.1) of the simulator and assembler that is mostly, but not entirely compatible with the 1.0 specification above, and only implements a few instructions and addressing modes. The text below repeats to some extent the things explained above. The examples may or may not work in further versions. However, the text characterizes the first release and might be useful to somebody.

*

*   *

Based on [1], we try to implement a simulator, not of the Felix (in the SIMH style), but of one single threaded Felix job that must consist of a restriction to the most commonly used instructions and of the ASSIRIS assembler, together with essential control instructions and macros. The job is described by an ASCII file (Felix used EBCDIC) on the standard input, resulting in a printout on the standard output. The input code is assembled in machine code that is probably the same with the Felix one, and executed through interpretation.

The simulator is currently only for Linux. It is written in Ada and published under GPLv2. (Incidentally, one of the designers of the SIRIS-3 operating system, Ichbiah, later went on to become the chief designer of the Ada-83 language.) It is a simple program, using only the standard libraries, it should be easy to port on other systems.

6.3.1 How to install

You must install the GNU NYU Ada Translator (gnat), the Ada component of the gcc compiler. On debian derived Linuxes use:

```
apt install gnat
```

Then, you unpack the distribution zip or tar.gz in a directory and give the command ‘make’.
To test a job say:

```
./felix <hello_world.ASSIRIS
```

...or `./felix -h` or felix anything to get an ’about’ and then a ’help’.
For more, you should really ’use the force’ (read the source).

6.3.2 The output of the ”. LIST ’HELP’ ” control card in version 0.1.1

The help is reproduced below:

```
GENERALITIES

This version only has 64K of memory and only knows some of the instructions:
AD4I BRU BCF BCT BAL CP1I CP1 CP2 CP4 EO2 EO4 EX2 EX4 LDC2 LDC4 LD1I LD2I LD1
LDL2 LDH2 LD4 LDM LD4I MG2 MG4 ST1 ST4 STH2 STM SB4I SB4
to which we added a couple more (see below): PRINT HALT
Only the direct and indirect addressing modes are implemented in this version.
The indirect addressing mode works for exactly one indirection.


CONTROL CARDS

The following cards (LIST and CONF) can appear anywhere in the input stream:
```

```
. LIST opt{,opt}* where opt can be:

     'HELP'     -- include this help text in the listing
     'ABOUT'    -- introduction to the FELIX/ASSIRIS system
     'SYMS'     -- the symbols table
     'LINKS'    -- the linkings performed by the link editor up to that point
     'DUMP'     -- `VIDAGE MEMOIRE' at that point
      MSG:'x'    -- display the message at that point

. CONF opt{,opt}* -- currently is ignored, it will provide for configuration of the system

The following cards can only appear in a specifc sequence.
The sequence is: JOB, COMPILE, LINK, RUN, EOJ; then, you may repeat.
COMPILE must be follwed by ASSIRIS (. COMPILE ASSIRIS)
RUN will admit one option, KINS:n where n is the number of thousands of
instructions to run. Without it, the simulator only runs 500 instructions
then stops (as was necessary in early tests).

Otherwise, you may add any options to the cards, but they are currently ignored.


DIRECTIVES

Directives are cards that can occur only between '. COMPILE ASSIRIS' and 'END'
The directives: ORG, EQU, DS, DB, ALIGN, work as expected. EQU defines a symbol.
ORG changes the address (that must be its argument) where the assembler generates code
DS is followed by a '-delimited string, the ASCII characters of which it puts into memory
in succesive locations

DB is followed by a sequence of byte-sized numbers (0..255), comma separated, that ar put into memory in sequ
the number can be: [-]ddd, decimal numbers; X'xx' hexadecimal, C'c' characters.

ALIGN is essential. Before assemblying machine code, the assembly address must be aligned to 4 bytes.
After DSs and DBs, always use ALIGN to synchronise the address to a 4-byte alignment, if code follows.
ALIGN take an optional argument about at what pace to align, in bytes. The default is 4.

END X must include this X which is the address, usually a label, where RUN will start execution.
CSECT is ignored.
Label expressions are not implemented, you can't say 'BRU ADDR+8' or something


NEW INSTRUCTIONS

HALT will finish running
PRINT,r will print the LSB of register r as an ascii character on stdout
PRINT,r C'd' will print the register r as a decimal on stdout
PRINT,r C'xy' will print the two ASCII characters x and y on stdout
```

Two test programs that work in V0.1.1 (and may not work exactly in future 0.x versions).

6.3.3 Hello world

```
. JOB HELLOWRL
. COMPILE ASSIRIS
```

```
         CSECT
STR      DS 'Hello World'
         DB 10,0
         ALIGN
HELLOW   LD4I,9 STR
DLOOP  LD1,4 *9,0
 CP1I,4 0
 BCT,8 NOPRINT
         PRINT,4
NOPRINT  AD4I,9 1
 CP1I,4 0
         BCF,8 DLOOP
 PRINT,3 C'..'
         PRINT,3 C'GA'
         PRINT,3 C'TA'
 HALT
         END HELLOW
. LINK
. RUN
. LIST MSG:'salut','SYMS','LINKS','DUMP'
. EOJ
```

6.3.4 Bogomips

```
. JOB BOGOMIPS
. COMPILE ASSIRIS
         CSECT
TESTDEC  LD4I,3 1000
 LD1I,4 10
DLOOP    LD4I,5 2500
 PRINT,2 C'..'
*        PRINT,3 C'd'
ILOOP    SB4I,5 1
         BCF,8 ILOOP
         SB4I,3 1
         BCF,8 DLOOP
 PRINT,3 C'..'
         PRINT,3 C'GA'
         PRINT,3 C'TA'
 HALT
         END TESTDEC
. RUN KINS:120000
. EOJ
```

7 The Felix-CC24 job simulator

This version, that is possible in the future, would attempt to simulate the segmented architecture of the felix-C-256, with addresses of up to 24 bits, but with segments of, at most, 64K (16 bits of address). Unlike the Felix-CC16, that is focused on simulating the instructions and the assembler, this is focused on simulating the operating system, including 'multiprogramming' [that is multithreading], the librarian and the SGF (*systéme de gestion de fichiers*, or file management system) but the OS (IRIS) simulator would not include a simulation of the peripherals at the hardware level, but system calls would be treated by the linux binary, using linux system entries, and files would be stored in the linux file system.

This version would to maintain an as close as possible resemblance to the user (assiris programmer) interface of the felix-assiris.

8 The Felix-CX24 job simulator

This is in fact a minimal extension of the CC16. It does *NOT* include sectioning, it is like a MC-68K with Assiris instructions. It only adds three instructions: (1) conditional relative jump, possibly over the next instruction (like the conditional short branch in MC68k); (2) long jump on immediate 24 bits addresses; a possible PREAL, load of a PC-relative address (Rr `<-` PC+D). The maximum length of 24 bits is otherwise imposed as in CC24. However, segments are not used in the system. Each program runs in a full 16M virtual memory space, and text and data of the program must fit in this space.

There is an extensive implementation of the HOST instruction.

9 The Felix-CX32 job simulator

The CX32 is a possible version, in the future, that extends the felix architecture in a broad way, for use as a virtual machine on modern computers, without significant concern to maintain compatibility.

There could be more registers and totally new instructions, including instruction formats. Flat addressing segments of any size (up to 2Gb or 4Gb) may be introduced.

Not all instructions in the Felix set would necessarily be supported, as the system calls could be quite different.

One way to extend the ISA might be with 64 bit or 96 bit instructions or even larger. One scheme could be to use two opcodes, 'HOP' and 'STOP'. When the (virtual) processor encounters a 'HOP', it could keep colecting words until the first STOP. O each word, 7 bits are the HOP/STOP opcodes, but the remaining 25 are usable to define a totally different 25 bit, 50 bit or 75 bit (or more) instruction set.

The primary extended instructions would be 32 bit immediate instructions in the 50 bit format. They could be:

The instruction would contain: one B register, one R register, opcode (8 bits), 32 bit address or value, 3 flags.

immediate 32 bit load, add, sub, mul, div

32 bit jump/conditional

32 bit call

A number of addressing modes..

References

[1] FELIX C 256–Structura si programarea calculatorului. Vasile Baltac, Ion Căruţaşu, Petru Macareie, Corneliu Maşek, Victor Megheşan, Maria Mocică, Lucia Popescu, Werner Schatz. Editura Tehnică, Bucureşti, 1974.

[2] Some Key Aspects in the History of Computing in Romania. Vasile Baltac, Horia Gligor. https://github.com/cronica-it/arhiva/releases/download/2015/vbaltac-some-key-aspects-in-the-history-of-computing-in-romania.pdf

[3] Introducere în sistemul de operare SIRIS. Horia Georgescu, Petre Preoteasa. Ed. Albatros, 1978.

[4] ASSIRIS, SGF şi implicaţiile lor în FORTRAN şi COBOL. Minerva Bocşa. Editura Facla, Timişoara, 1986.

[5] Limbaje de programare, ASSIRIS Manual pentru licee de matematică-fizică, profilul matematică-informatică, clasa a XII-a. Mihai Jitaru, Alexandru Teodorescu. Editura didactică şi pedagogică, Bucureşti 1978.

[6] Memento (principalele instrucţiuni, formate şi comenzi) Felix-C-256, scanat la: https://cronica-it.github.io/arhiva/assets/1975/babesbalyai-memento-felix-c-256.pdf

[7] Elemente ale sistemului de operare Siris 3. Ştefan Măruşter. Editura Facla, Timişoara, 1980.

[8] Programarea în limbaje de asamblare ASSIRIS. E. Munteanu, V. Corstea, M. Mitrov, Ed. Tehnică, Bucureşti, 1976.

[9] Xerox Meta-Symbol. Sigma 5–9 computers. Language and Operations. Reference Manual. https://bitsavers.computerhistory.org/pdf/sds/sigma/lang/900952G_metaSymbolLangRef_Oct75.pdf