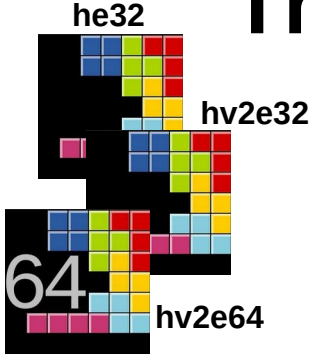
One Step Beyond ...

or

The eXtended Jack (XJack) Programming Language & its Eco System



What are we talking about?

Frontend:
High level language

Intermediate Language (IL)*

or

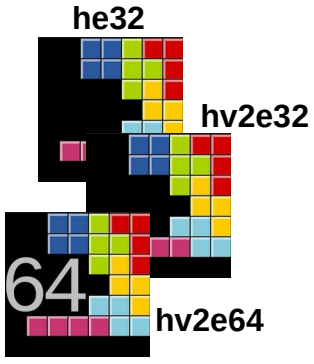
Virtual Machine Code (VM)**

Backend:
Low level language
(Native target platform language)

Examples:
- C/C++/C#
- Java
...
- Jack/XJack

Examples:
- LLVM
- Java VM
...
- Jack/XJack IL

Examples:
- ARM assembly language
- x86 assembly language
- RISC V assembly language
...
- Hack v1/v2



The tools of a toolchain!

continued ...

* **Intermediate Language: A textual representation**
** **Virtual Machine Code: A byte code / binary representation (also: Intermediate representation IR)**

What are we talking about?

Backend:
Low level language
(Native target platform language)

Example:
Hack v1/v2

Native target platform
machine code (MC)

Example:
Hack v1/v2 platform

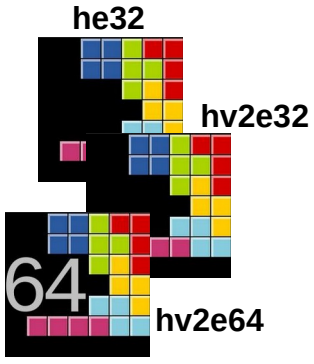
BIOS
or
Firmware
or
Operating System (OS)

Example:
HV2OS

Platform Executables
(`binaries`, `programs`, `apps`)

Example:
Pong, CGDGame etc.

The tools of a toolchain!



Compiler Building Tools

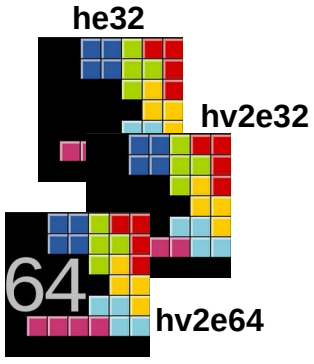
In the old days (say: 80s): lex & yacc

A little younger (say: 90s): flex & bison (GNU versions)

lots & lots of others (lesser well known …)

These days (probably): ANTLR (4)

Now, forget about these ...

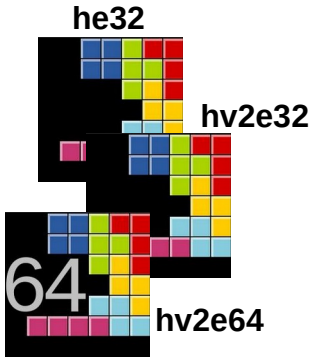


IL/VM code

Intermediate languages or virtual machine codes aren't new!

- (Pascal) p-code (since early 80s)
- java bytecode (since mid 90s)
- LLVM (since ~2000)
- BitTorrent (since 2001)
- Gimple (since ~2002)
- .NET MSIL (since 2002)
- Ethereum VM (recent dev)

lots & lots of others ...

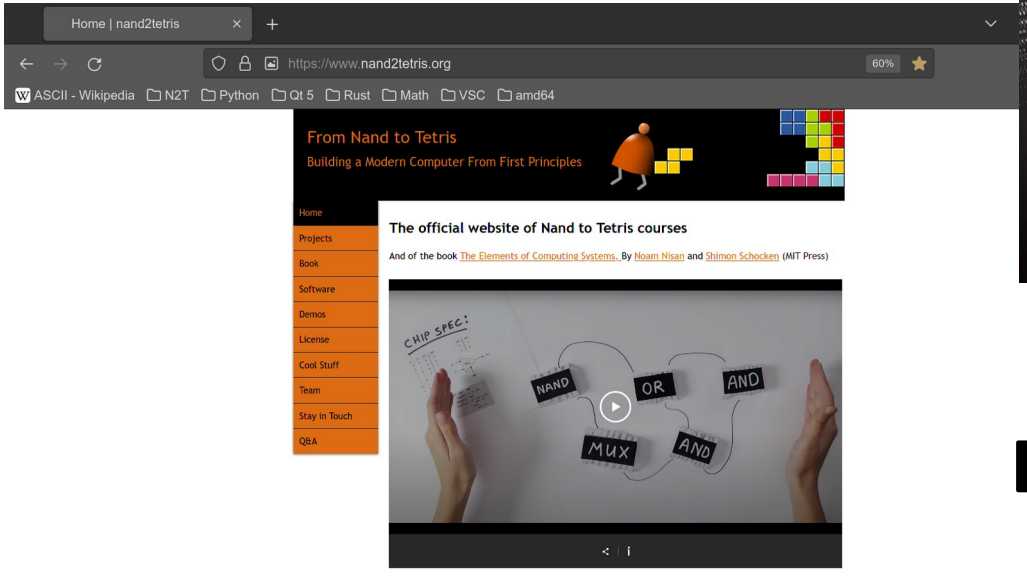


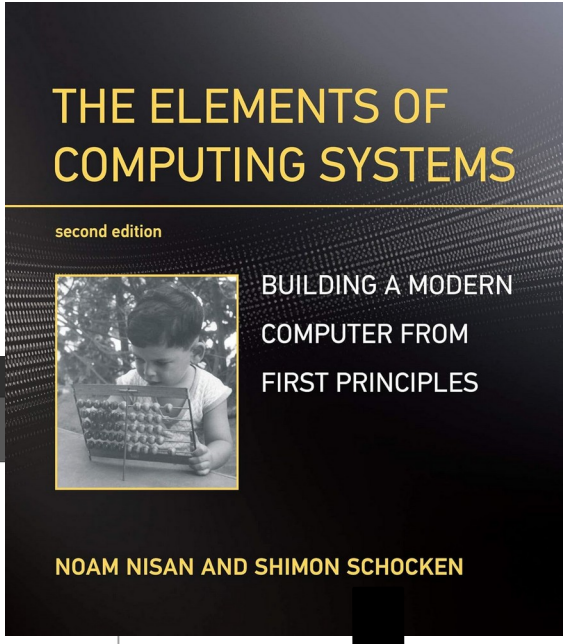
Genesis of a toolchain

In the beginning, there was a book:

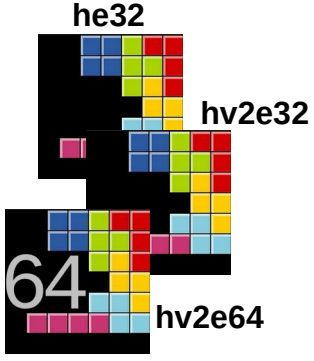
and a computer science course:

https://www.nand2tetris.org/





The ‚**Hack**‘ platform for the ‚**Jack**‘ programming language

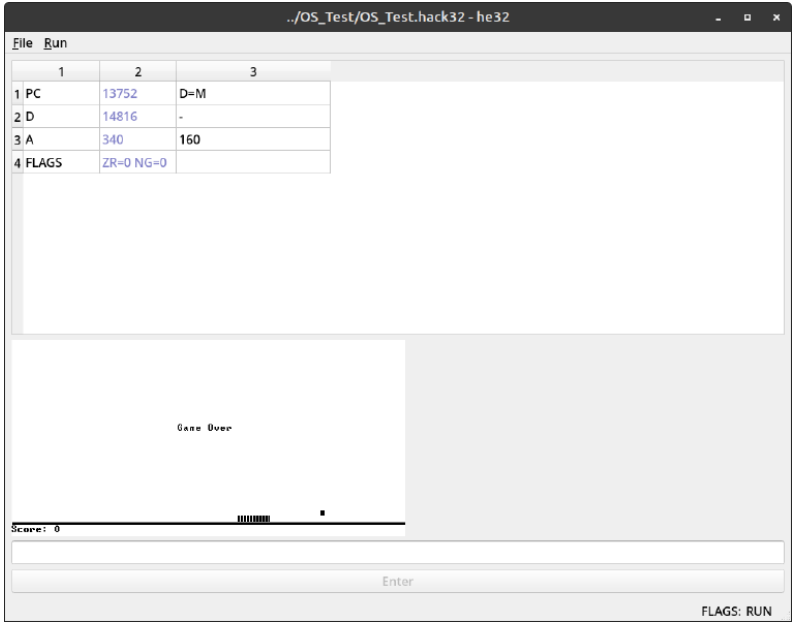


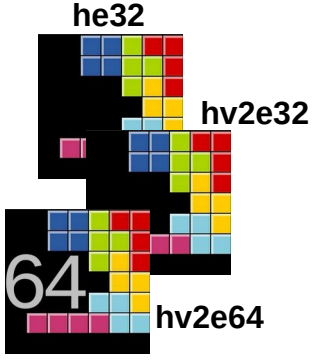
Genesis of a toolchain

After course completion, an emulator became necessary,

the hack 32-bit emulator:

he32

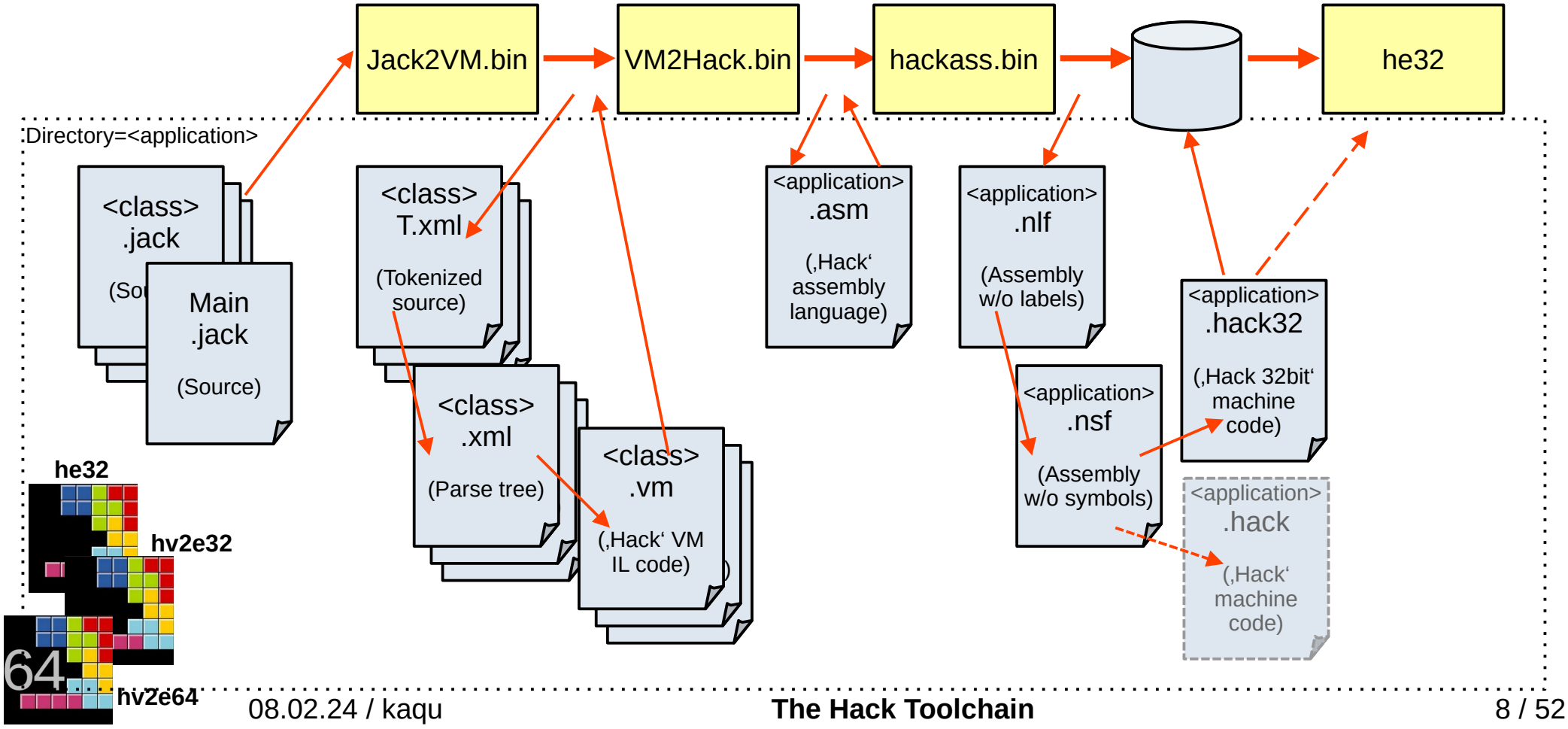




(see https://www.hacknology.de/projekt/2022/he32/ for details)

Genesis of a toolchain

The ‚Hack' toolchain for **he32**:



Genesis of a toolchain

```
/**
* Bounces off the current wall: sets the new destination
* of the ball according to the ball's angle and the given
* bouncing direction (-1/0/1=left/center/right or up/center/down).
*/
method void bounce(int bouncingDirection) {
        var int newx, newy, divLengthx, divLengthy, factor;

        // dividing by 10 first since results are too big
        let divLengthx = lengthx / 10;
        let divLengthy = lengthy / 10;
        if (bouncingDirection = 0) {
                let factor = 10;
        }
        else {
                if (((~(lengthx < 0)) & (bouncingDirection = 1)) | ((lengthx < 0) & (bouncingDirection
                        let factor = 20; // bounce direction is in ball direction
                }
                else {
                        let factor = 5;
                } // bounce direction is against ball direction
        }

        if (wall = 1) {
                let newx = 506;
                let newy = (divLengthy * (-50)) / divLengthx;
                let newy = y + (newy * factor);
        }
        else {
                if (wall = 2) {
                        let newx = 0;
```

he32

Directory=<application>

<class>
.jack

(Source)

he32

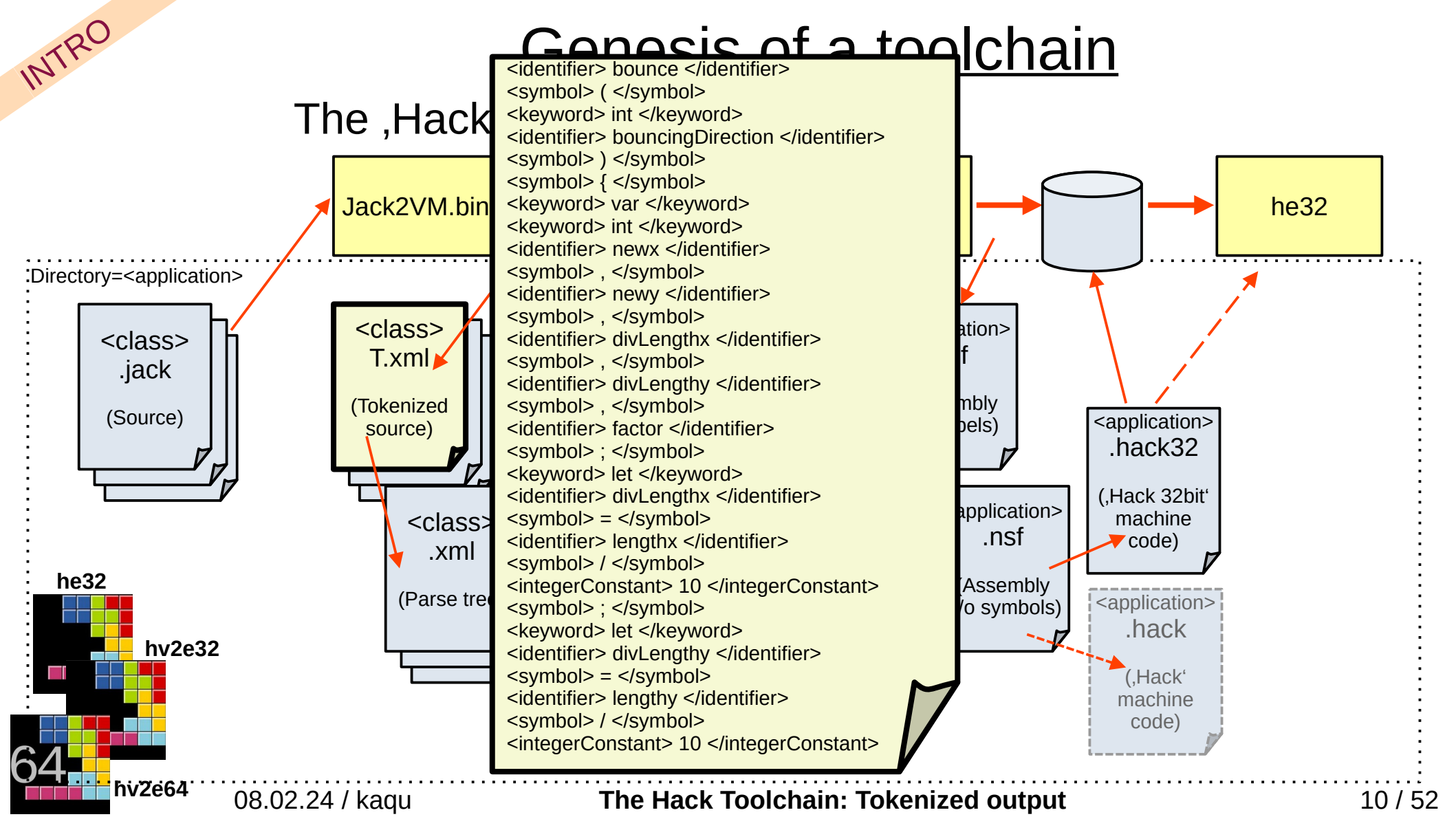
hv2e32

64

hv2e64

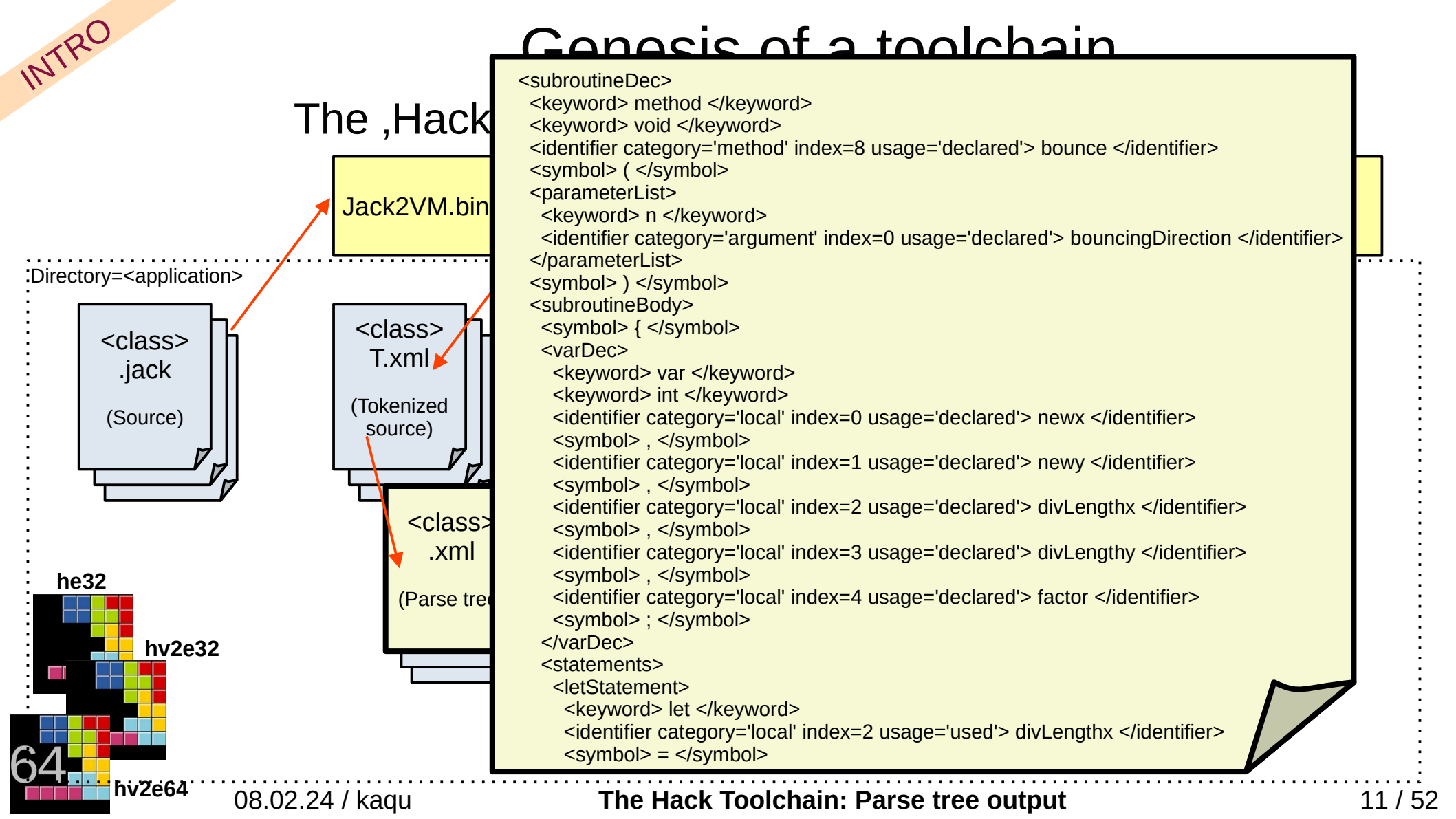
Genesis of a toolchain

The ‚Hack



Genesis of a toolchain

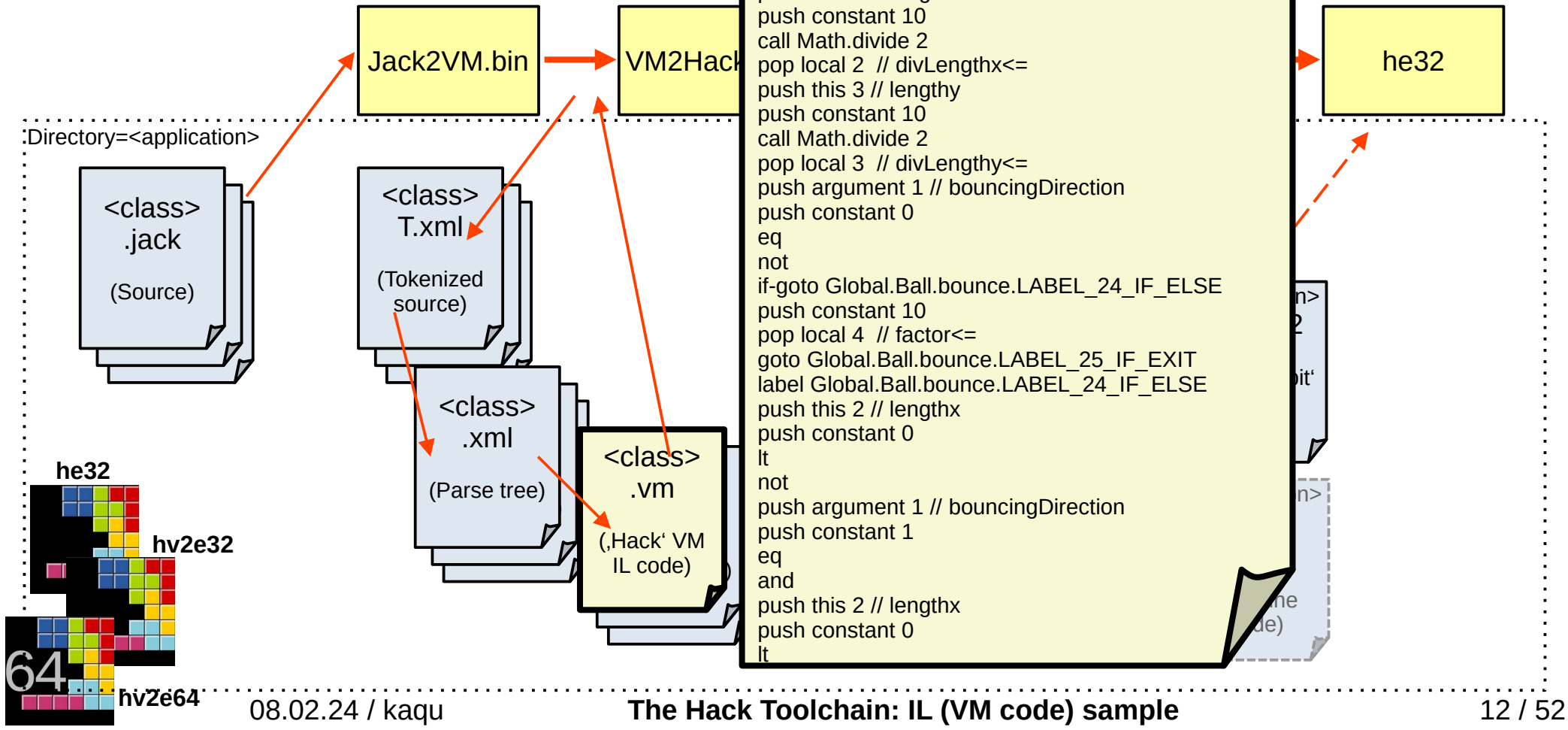
The ‚Hack




```
<subroutineDec>
  <keyword> method </keyword>
  <keyword> void </keyword>
  <identifier category='method' index=8 usage='declared'> bounce </identifier>
  <symbol> ( </symbol>
  <parameterList>
    <keyword> n </keyword>
    <identifier category='argument' index=0 usage='declared'> bouncingDirection </identifier>
  </parameterList>
  <symbol> ) </symbol>
  <subroutineBody>
    <symbol> { </symbol>
    <varDec>
      <keyword> var </keyword>
      <keyword> int </keyword>
      <identifier category='local' index=0 usage='declared'> newx </identifier>
      <symbol> , </symbol>
      <identifier category='local' index=1 usage='declared'> newy </identifier>
      <symbol> , </symbol>
      <identifier category='local' index=2 usage='declared'> divLengthx </identifier>
      <symbol> , </symbol>
      <identifier category='local' index=3 usage='declared'> divLengthy </identifier>
      <symbol> , </symbol>
      <identifier category='local' index=4 usage='declared'> factor </identifier>
      <symbol> ; </symbol>
    </varDec>
    <statements>
      <letStatement>
        <keyword> let </keyword>
        <identifier category='local' index=2 usage='used'> divLengthx </identifier>
        <symbol> = </symbol>
```

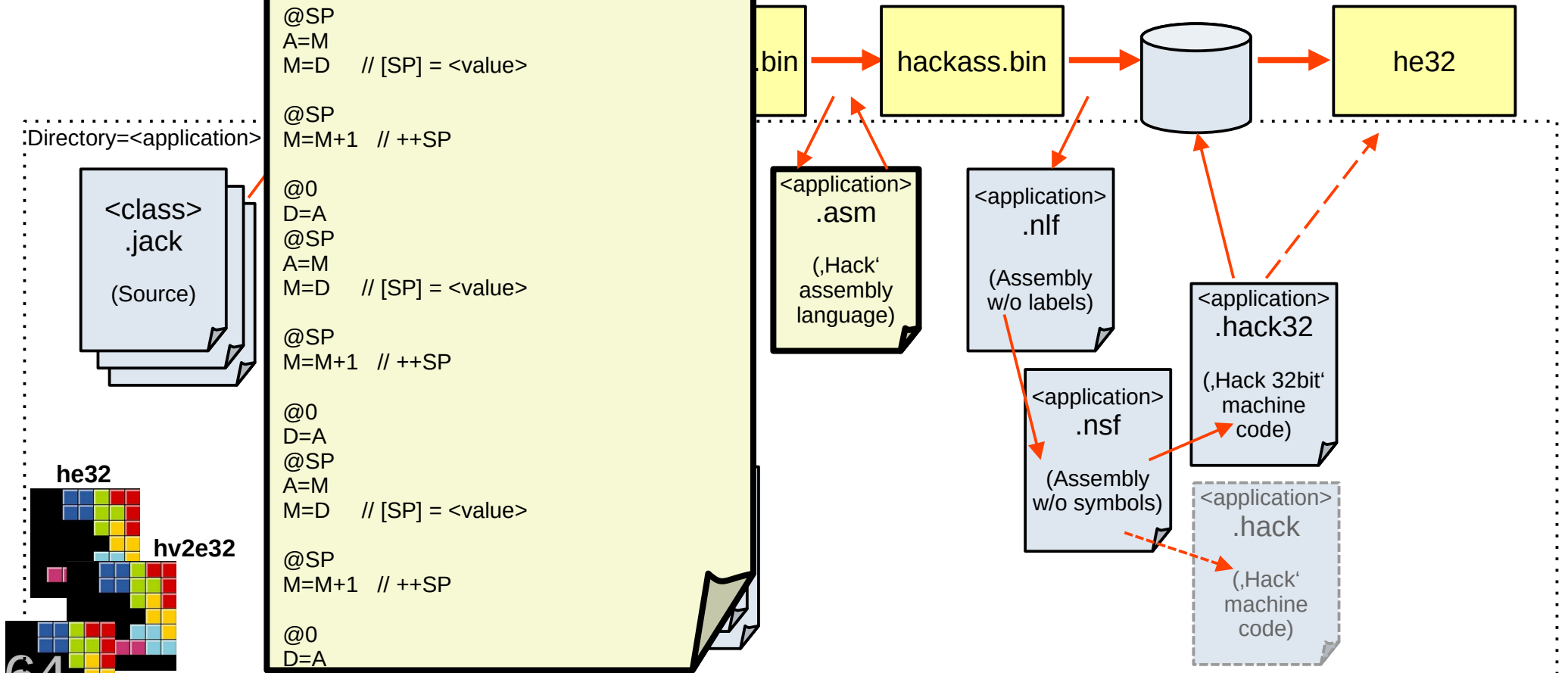
Genesis of a toolchain

The ‚Hack' toolchain



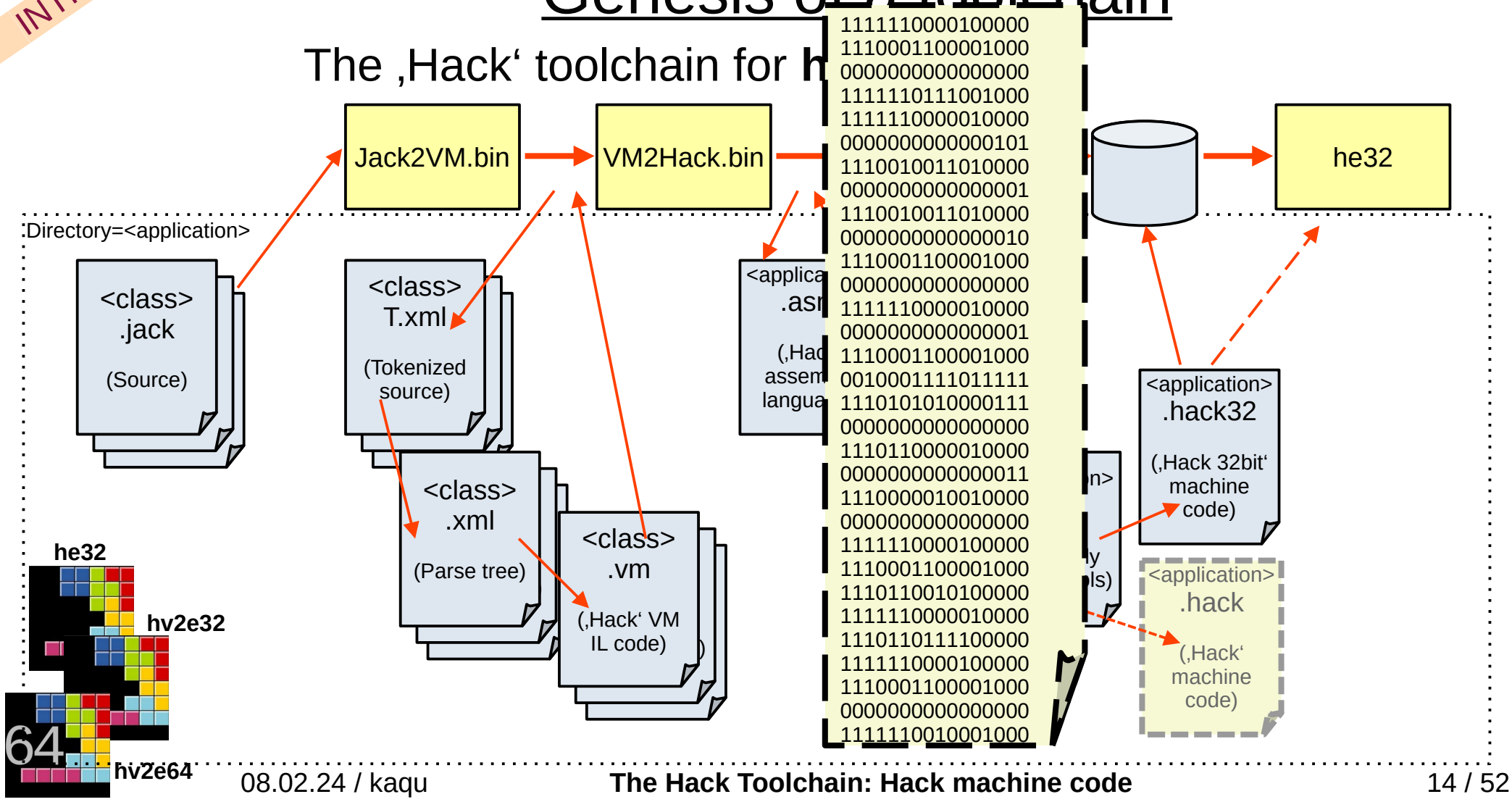
Genesis of a toolchain for **he32**:

```
(Ball.bounce)    // Ball.bounce entry point
                 // using 5 local variables
@0
D=A
@SP
A=M
M=D     // [SP] = <value>

@SP
M=M+1   // ++SP

@0
D=A
@SP
A=M
M=D     // [SP] = <value>

@SP
M=M+1   // ++SP

@0
D=A
@SP
A=M
M=D     // [SP] = <value>

@SP
M=M+1   // ++SP

@0
D=A
```



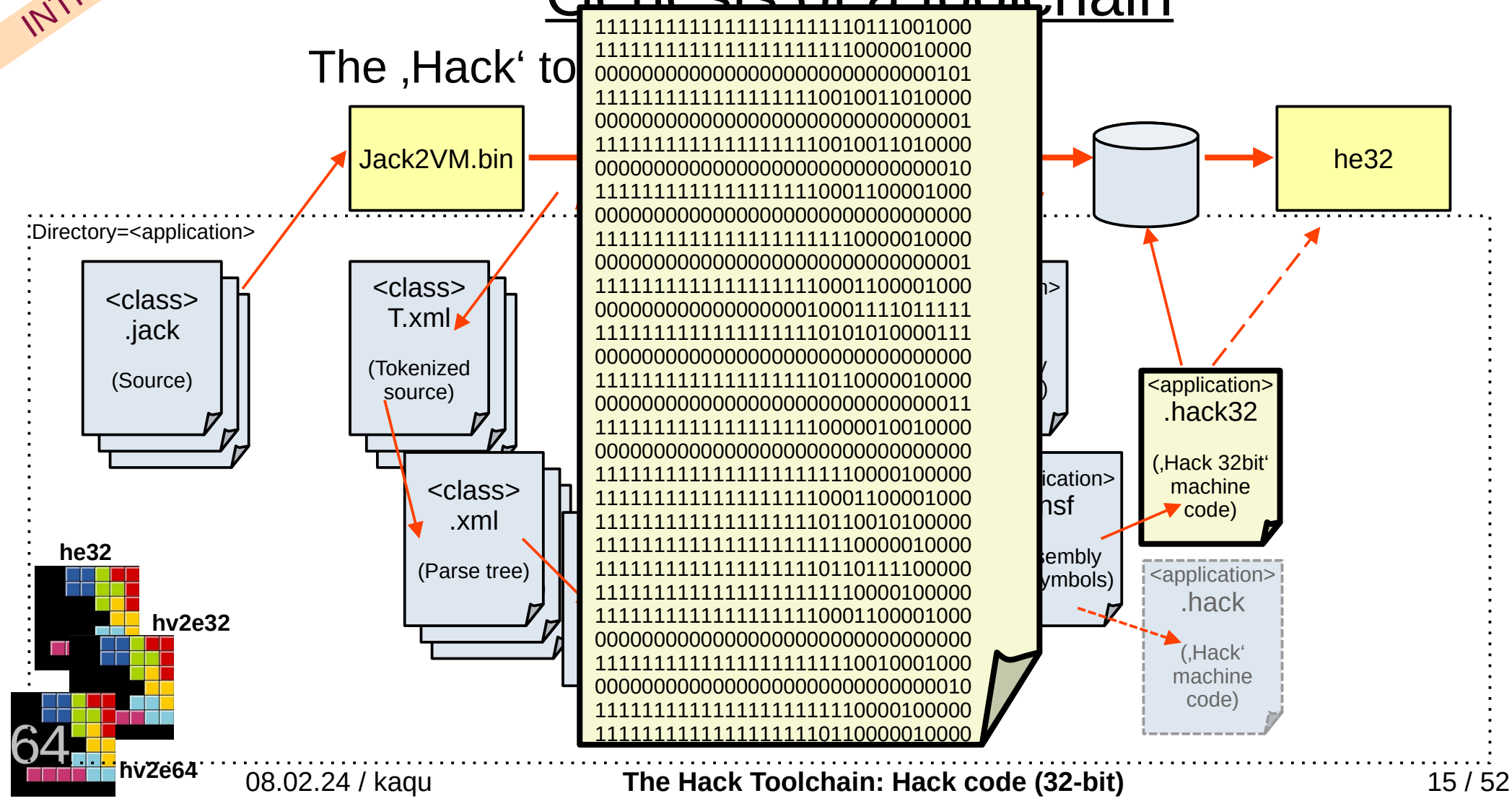
Genesis of a toolchain

The ‚Hack' toolchain for h



Genesis of a toolchain

The ‚Hack' to



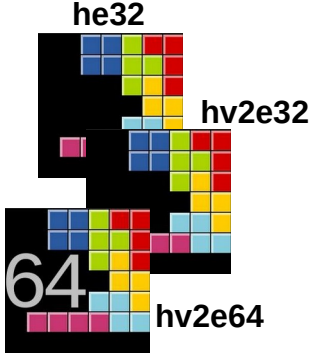
Genesis of a toolchain

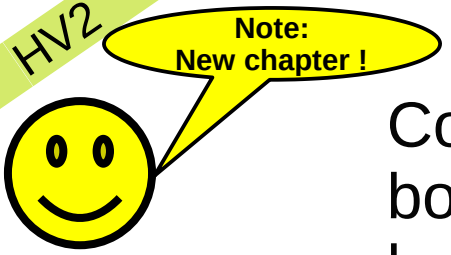
So we've added one platform to the zoo:

- x86 (w/ many many modes & extensions ...)
- ARM (also coming in several flavours …)
- RISC V (an evolution, 5th loop, maybe this time ...)
- PowerPC (don't know / never heard of ...)
- Sparc, ia64 (these parrots are dead!)
- Mips (this one – almost - as well ...)
- Xtensa (smart ideas, probably phased out next ...)

etc. (lots of others, long forgotten ...)

- **Hack (‚v1', a course development, superfluous as any!)**

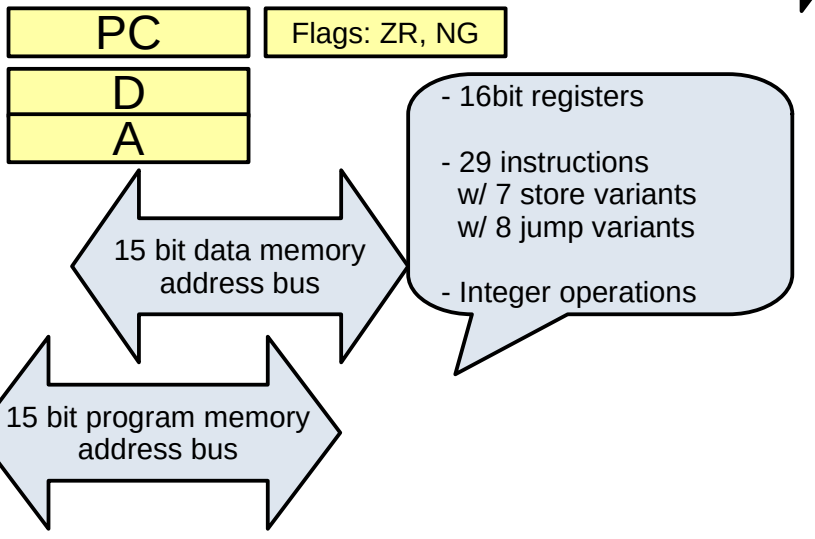




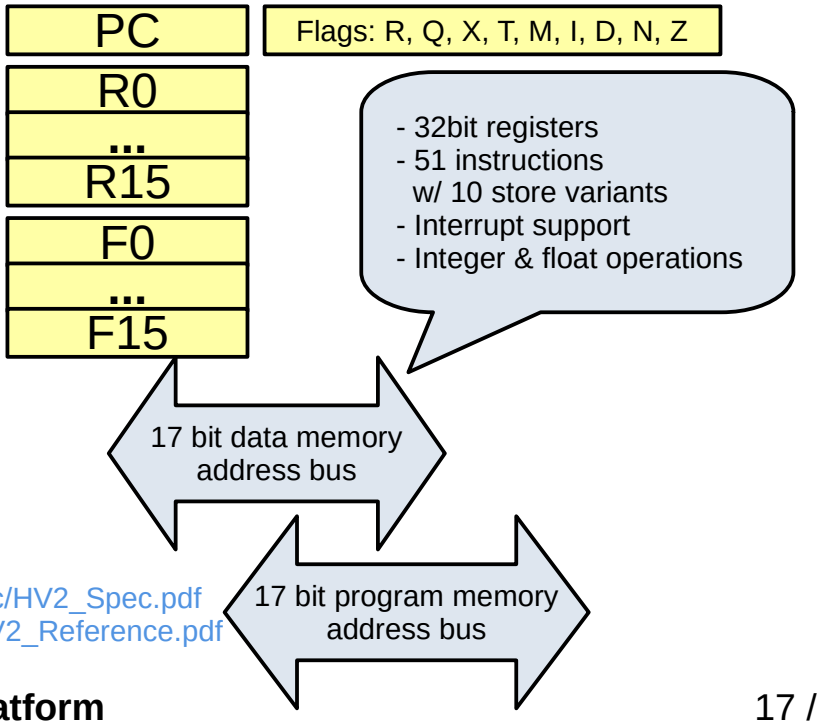
Genesis of a toolchain

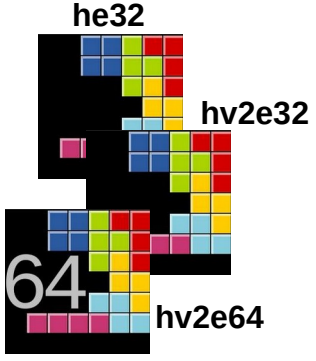
Coming from the courses ‚Hack' platform (Part 1 of the book, chip design), a new platform ‚Hack Version 2' has been proposed.

Hack:



Hack v2*:





* see https://git.hacknology.de/kaqu/hv2e32/src/branch/master/doc/HV2_Spec.pdf & https://git.hacknology.de/kaqu/hv2e32/src/branch/master/doc/HV2_Reference.pdf for details

Genesis of a toolchain

Instruction set & mnemonic changes ...

Hack:

A instruction

Symbolic: @*xxx*

(*xxx* is a decimal value ranging from 0 to 32767, or a symbol bound to such a decimal value)

***variants:**
direct/indirect source & destination
w/ or w/o
increment/decrement or offset

Also: Register renaming
R0→SP, R1→LCL, R2→ARG etc.

Examples:
L SP, #0 //Load register direct w/ const.
S [SP], #0 //Store const. indirect
L SP, TEMP1+#0 // Load register w/ register + const.
…
S [SP++], [TEMP2 + #8] // Store indirect source register w/ post increment
// from indirect w/ offset

SCREEN 16384
KBD 24576

D+A	D+M	0	0	0	0	1	0
D−A	D−M	0	1	0	0	1	1
A−D	M−D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1

$a == 0$ $a == 1$

JGE	0	1	1	if $comp \geq 0$ jump
JLT	1	0	0	if $comp < 0$ jump
JNE	1	0	1	if $comp \neq 0$ jump
JLE	1	1	0	if $comp \leq 0$ jump
JMP	1	1	1	unconditional jump

Hack v2:

Load/Store (variants*):
L, S,
FL, FLCONST, FS,

ALU Ops (variants*):
NOT, NEG, FNEG, INC, FINC, DEC, FDEC, ADD, FADD, SUB, FSUB, AND, OR, MUL, FMUL, DIV, FDIV, FTOI, FFROMI

Jumps (variants*):
JGT, JEQ, JGE, JLT, JNE, JLE, JMP,
FJGT, FJEQ, FJGE, FJLT, FJNE, FJLE

Interrupt:
ICALL, IRET,

Other (variants*):
JUMP, SETFLAG , S<cond>, RETURN, CALL, LCONTROL, CONTROL, RCALL, SROM, SEMA, (ICPL)

Pseudo-Ops (variants*):
CLR,

NOTTOS, NEGTOS, INCTOS, DECTOS, ADDTOS, SUBTOS, ANDTOS, ORTOS, MULTOS, DIVTOS,
FNEGTOS, FINCTOS, FDECTOS, FADDTOS, FSUBTOS, FMULTOS, FDIVTOS

PUSH <variants>, POP <variants>,
FPUSH, FPOP,

HALT, BKPT, WAIT

he32

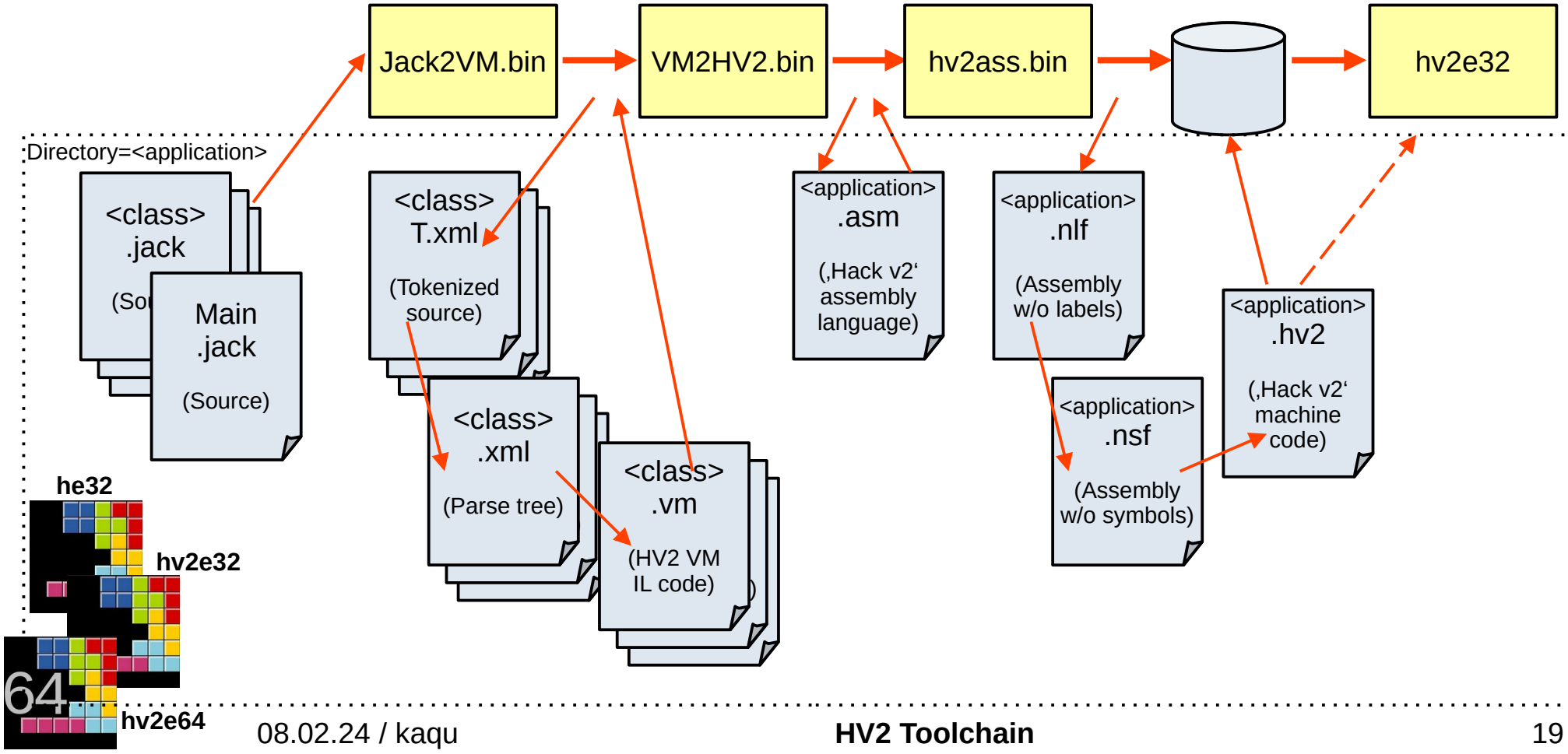
etc.

64

hv2e64

Genesis of a toolchain

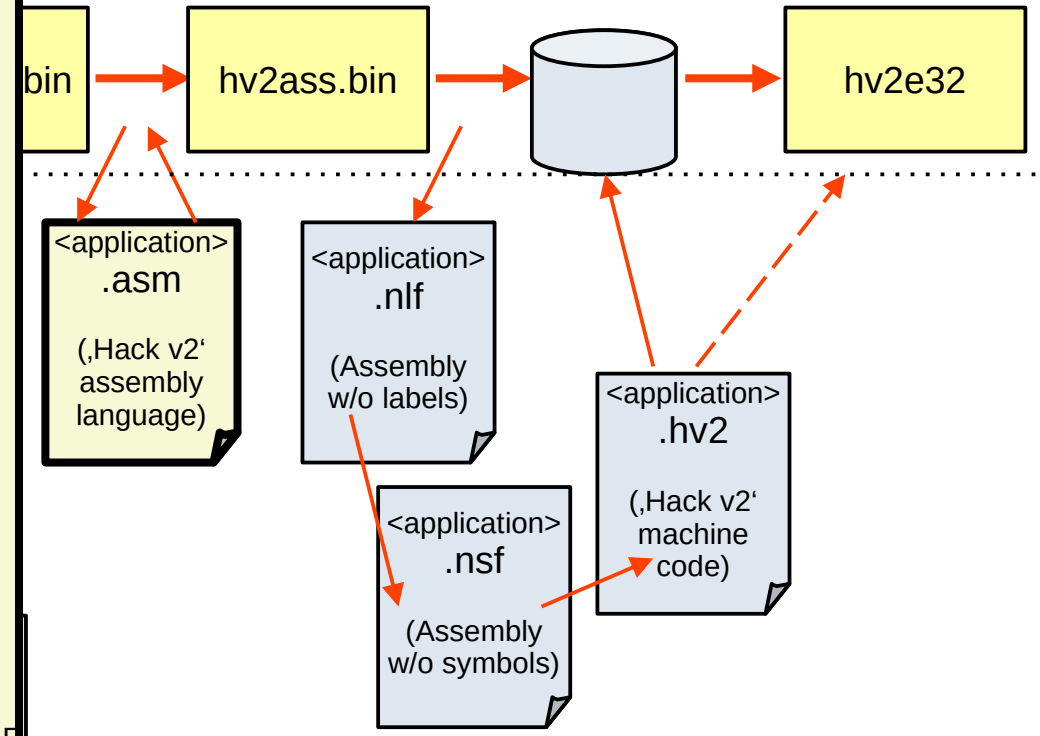
The ‚Hack v2' toolchain for **hv2e32**:



Genesis of a toolchain

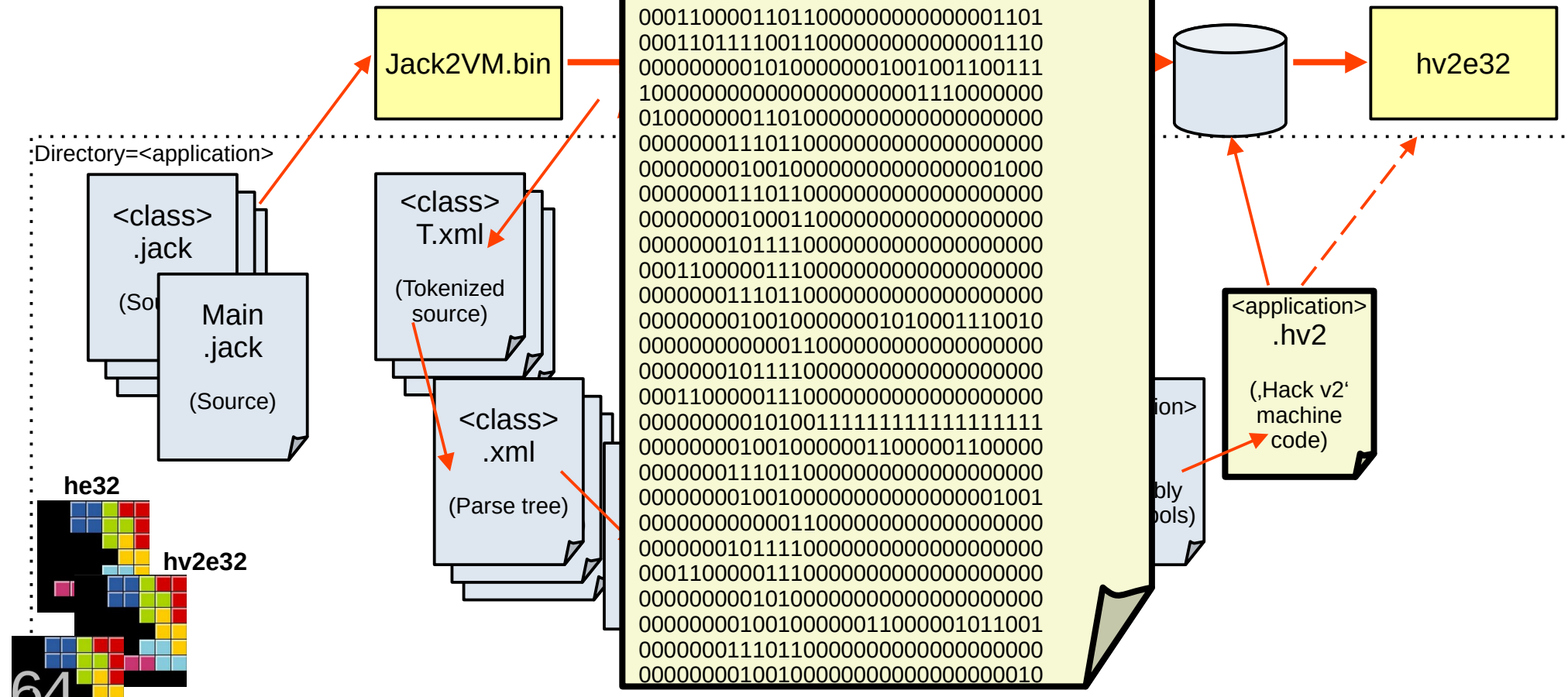
ain for **hv2e32**:

```
@LINE#223#jack/apps/Pong/Ball.jack#:
Ball.bounce:
L SP, SP + #5
PUSH [ARG + #0]
POP THIS
@LINE#227#jack/apps/Pong/Ball.jack#:
PUSH [THIS + #2]
PUSH #10
POP TEMP1
S [ZERO], TEMP1
DIVTOS [ZERO]
DEC SP
S [LCL + #2], [SP]
@LINE#228#jack/apps/Pong/Ball.jack#:
PUSH [THIS + #3]
PUSH #10
POP TEMP1
S [ZERO], TEMP1
DIVTOS [ZERO]
DEC SP
S [LCL + #3], [SP]
@LINE#229#jack/apps/Pong/Ball.jack#:
PUSH [ARG + #1]
PUSH #0
DEC SP
SUBTOS [SP]
JUMPNE [--SP], #Ball.Ball.bounce$Global.Ball.bounce.LABEL_26_IF_E
@LINE#229#jack/apps/Pong/Ball.jack#:
PUSH #10
DEC SP
S [LCL + #4], [SP]
```



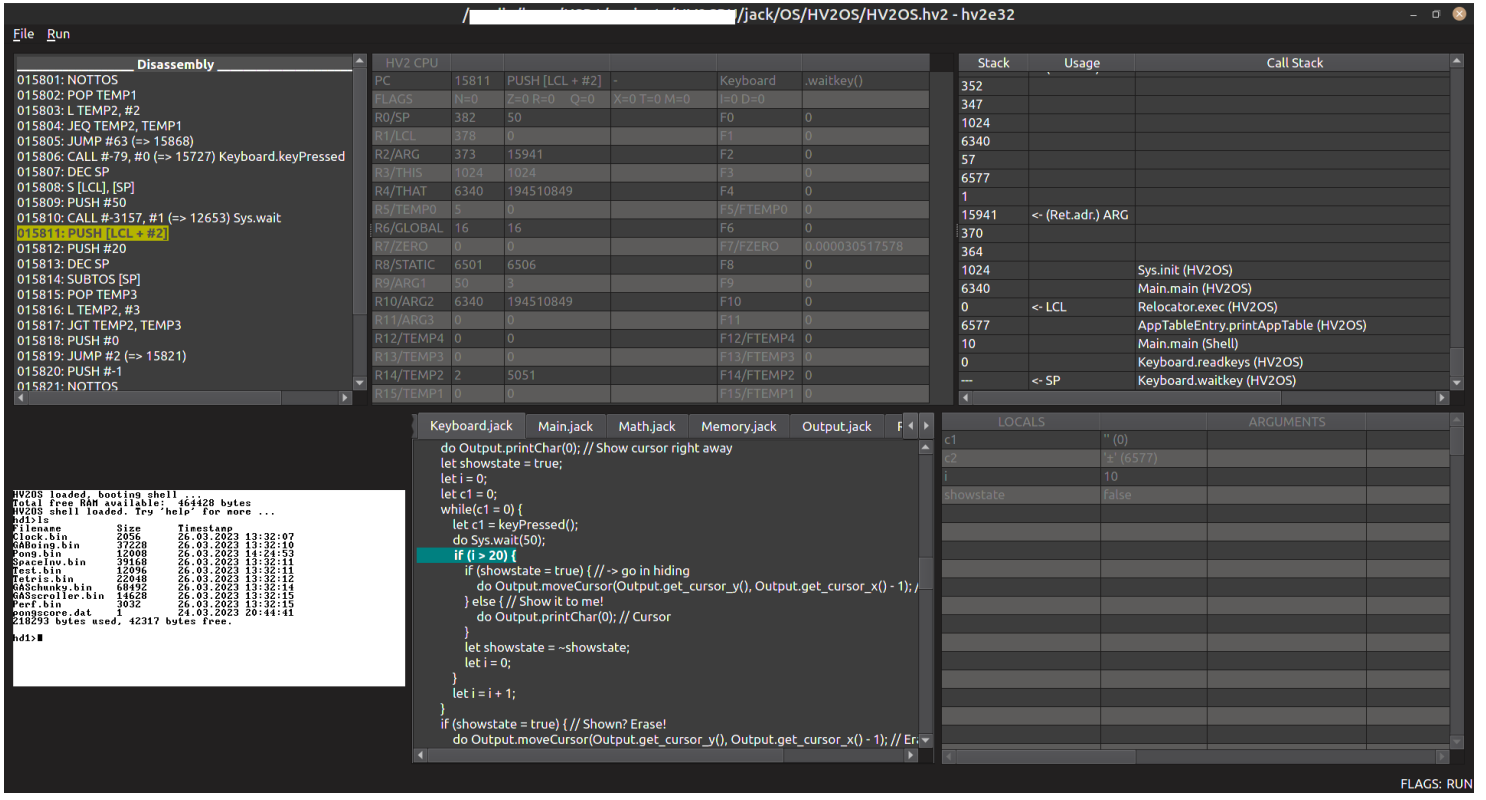
Genesis of a toolchain

The ‚Hack v2

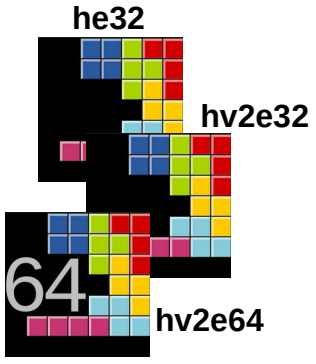


Genesis of a toolchain

The ‚Hack V2' emulator **hv2e32**:



(see https://git.hacknology.de/kaqu/hv2e32 for details)



/[redacted]/jack/OS/HV2OS/HV2OS.hv2 - hv2e32

File Run

Disassembly

```
015801: NOTTOS
015802: POP TEMP1
015803: L TEMP2, #2
015804: JEQ TEMP2, TEMP1
015805: JUMP #63 (=> 15868)
015806: CALL #-79, #0 (=> 15727) Keyboard.keyPressed
015807: DEC SP
015808: S [LCL], [SP]
015809: PUSH #50
015810: CALL #-3157, #1 (=> 12653) Sys.wait
015811: PUSH [LCL + #2]
015812: PUSH #20
015813: DEC SP
015814: SUBTOS [SP]
015815: POP TEMP3
015816: L TEMP2, #3
015817: JGT TEMP2, TEMP3
015818: PUSH #0
015819: JUMP #2 (=> 15821)
015820: PUSH #-1
015821: NOTTOS
```

HV2 CPU					
PC	15811	PUSH [LCL + #2]	-	Keyboard	.waitkey()
FLAGS	N=0	Z=0 R=0 Q=0	X=0 T=0 M=0	I=0 D=0	
R0/SP	382	50		F0	0
R1/LCL	378	0		F1	0
R2/ARG	373	15941		F2	0
R3/THIS	1024	1024		F3	0
R4/THAT	6340	194510849		F4	0
R5/TEMP0	5	0		F5/FTEMP0	0
R6/GLOBAL	16	16		F6	0
R7/ZERO	0	0		F7/FZERO	0.000030517578
R8/STATIC	6501	6506		F8	0
R9/ARG1	50	3		F9	0
R10/ARG2	6340	194510849		F10	0
R11/ARG3	0	0		F11	0
R12/TEMP4	0	0		F12/FTEMP4	0
R13/TEMP3	0	0		F13/FTEMP3	0
R14/TEMP2	2	5051		F14/FTEMP2	0
R15/TEMP1	0	0		F15/FTEMP1	0

Stack	Usage	Call Stack
352		
347		
1024		
6340		
57		
6577		
1		
15941	<- (Ret.adr.) ARG	
370		
364		
1024		Sys.init (HV2OS)
6340		Main.main (HV2OS)
0	<- LCL	Relocator.exec (HV2OS)
6577		AppTableEntry.printAppTable (HV2OS)
10		Main.main (Shell)
0		Keyboard.readkeys (HV2OS)
---	<- SP	Keyboard.waitkey (HV2OS)

```
HV2OS loaded, booting shell ...
Total free RAM available:  464428 bytes
HV2OS shell loaded. Try 'help' for more ...
hd1>ls
Filename        Size      Timestamp
Clock.bin       2056      26.03.2023 13:32:07
GABoing.bin     37228     26.03.2023 13:32:10
Pong.bin        12008     26.03.2023 14:24:53
SpaceInv.bin    39168     26.03.2023 13:32:11
Test.bin        12096     26.03.2023 13:32:11
Tetris.bin      22048     26.03.2023 13:32:12
GASchunky.bin   68492     26.03.2023 13:32:14
GAScroller.bin  14628     26.03.2023 13:32:15
Perf.bin        3032      26.03.2023 13:32:15
pongscore.dat   1         24.03.2023 20:44:41
218293 bytes used, 42317 bytes free.

hd1>
```

Keyboard.jack | Main.jack | Math.jack | Memory.jack | Output.jack | P

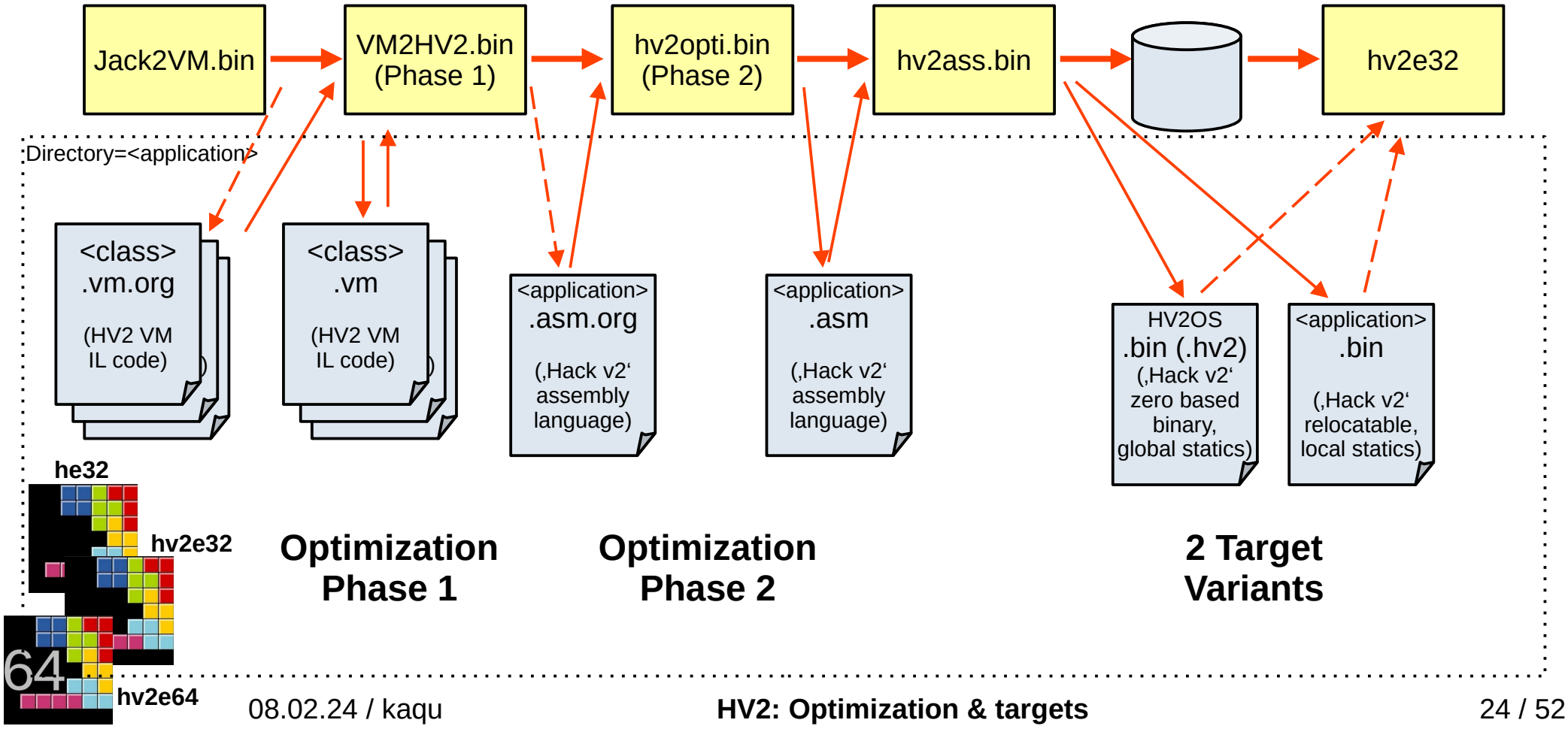
```
    do Output.printChar(0); // Show cursor right away
    let showstate = true;
    let i = 0;
    let c1 = 0;
    while(c1 = 0) {
      let c1 = keyPressed();
      do Sys.wait(50);
      if (i > 20) {
        if (showstate = true) { // -> go in hiding
          do Output.moveCursor(Output.get_cursor_y(), Output.get_cursor_x() - 1); /
        } else { // Show it to me!
          do Output.printChar(0); // Cursor
        }
        let showstate = ~showstate;
        let i = 0;
      }
      let i = i + 1;
    }
    if (showstate = true) { // Shown? Erase!
      do Output.moveCursor(Output.get_cursor_y(), Output.get_cursor_x() - 1); // Er
```

LOCALS		ARGUMENTS
c1	'' (0)	
c2	'±' (6577)	
i	10	
showstate	false	

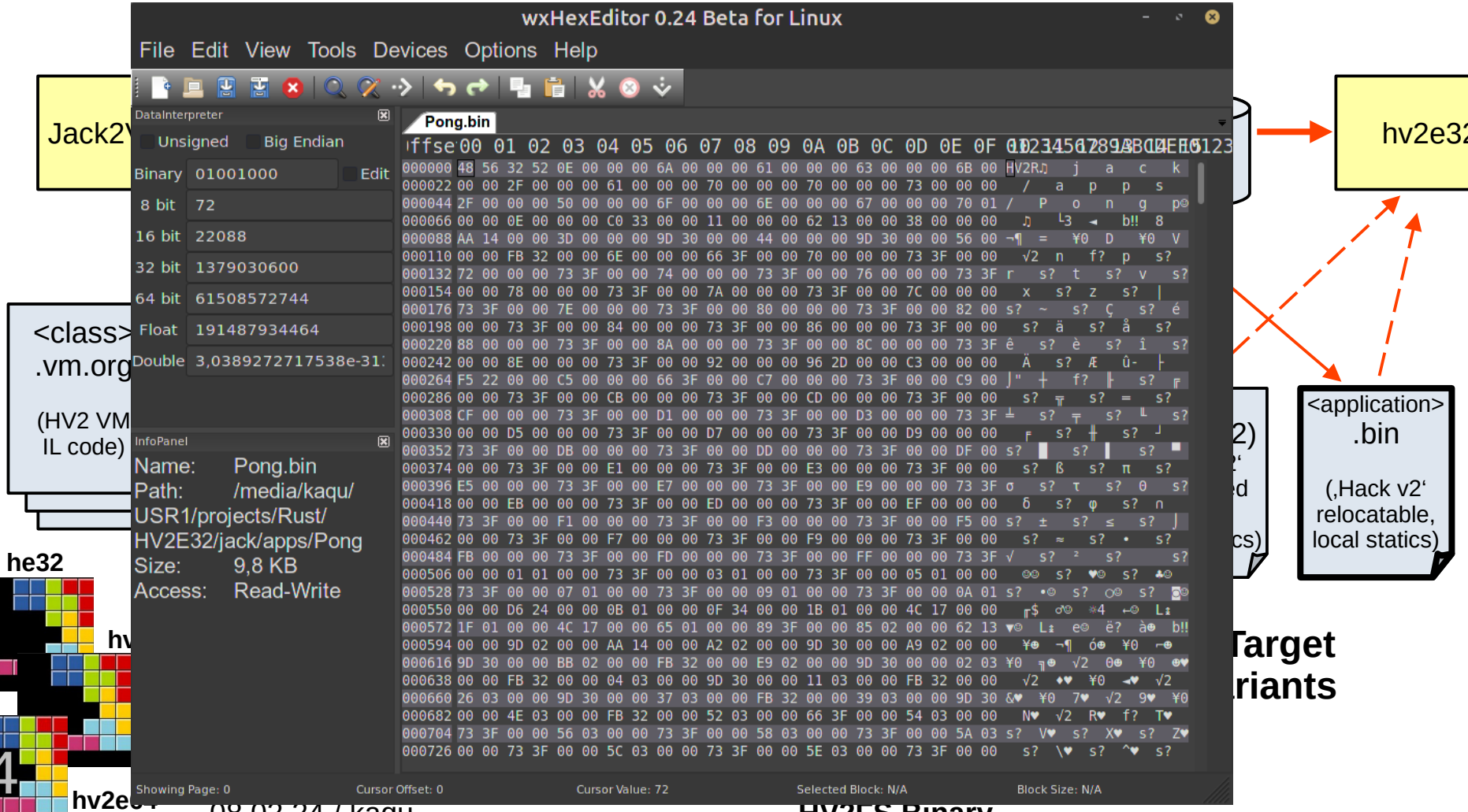
FLAGS: RUN

Genesis of a toolchain

The ‚Hack v2' toolchain for **hv2e32**:



Genesis of a toolchain



HV2

Language Extensions

The intermediate (VM) language had to be modified:

‚Hack‘ IL

Predefined constants: true, false, this, null

: local, argument, this, that, constant,
static, temp, pointer

pop <n>
push <n>

add, sub, neg, eq, get, lt, and, or, not

label <labelname>
goto <labelname>
if-goto <labelname>

function <functionname> <n_args>
call <functionname> <n_args>
return

‘Hack v2‘ IL

Predefined constants: true, false, this, null

: local, argument, this, that, constant,
static**|global**, temp, pointer, **fconstant**

pop <n>, **fpop <n>**
push <n>, **fpush <n>**

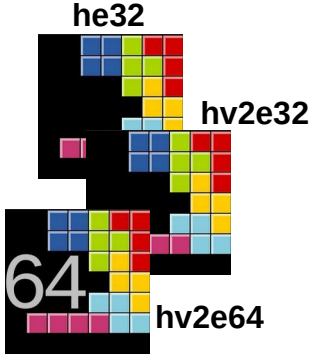
add, sub, neg, eq, gt, lt, and, or, not,
fadd, fsub, fneg, feq, fgt, flt

label <labelname>
goto <labelname>
if-goto <labelname>, **if-eq-not-goto <labelname>, if-lt-not-goto, if-gt-not-goto**

function <functionname> <n_args>
call <functionname> <n_args>
pcall <varname> <n_args>
return **<0|1>**

asm {<inline assembly code>;}+

attribute <public|private>



Quick recap grammar notation

Grammars consist of ‚terminals' & (non-terminal) ‚production' rules.

A terminal is a non-changeable symbol, non-terminals may (rather: will) be modified by the rules.

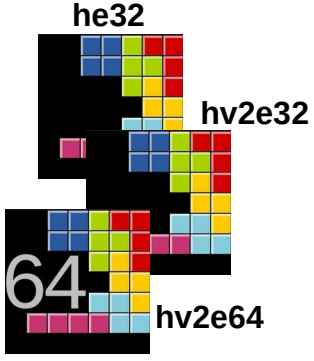
Left-hand side of a production rule is always a single non-terminal?
→ context free grammar → most programming languages

EBNF (Extended Backus-Naur form):

Probably the most common notation for describing grammars for programming languages. Includes regular expression syntax & simplified syntax (as opposed to the original BNF).

‚|'	:	Selection, permits alternative ‚productions'
'*'	:	Repetition, means **zero** or more times
'+'	:	Existance, means **one** or more times
'?'	:	Option, **zero** or only **once**
'(, ,)'	:	Grouping, so the previous symbols can be applied to a group of symbols
'[, ,]'	:	Optional symbols

… some others ...



Examples:

```
flt1  = [-|+]?[0-9]+`.`[0-9]          // Match a float number like 3.1415 etc.
flt2  = [-|+]?[0-9][0-9]?[0-9]?[0-9]?[0-9]?[0-9]?[0-9]?`.`[0-9]?[0-9]?[0-9]?[0-9]?[
```

HV2

Language extensions

The ‚Jack' programming language has been ‚eXtended' (hence ‚XJack', extensions in **bold**) in 4 categories:

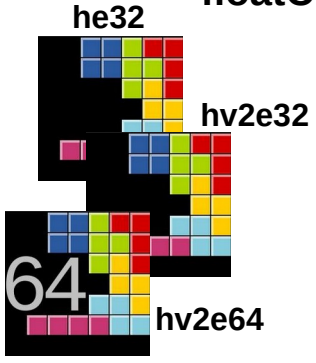
1. Lexical elements:

Keyword: `class` | `**public` | `private**` | `constructor' | `function` | `method` | `field` | `static` | `var` | `int` | `char` | `boolean` | `**float**` | `void` | `true` | `false` | `null` | `this` | `let` | `do` | `if` | `else` | `while` | `return` | `**for` | `switch` | `continue` | `break` | `asm**`

symbol: `{` | `}` | `(` | `)` | `[` | `]` | `.` | `,` | `;` | `+` | `-` | `*` | `/` | `&` | `|` | `<` | `>` | `=` | `~`

integerConstant: a decimal**/hex/binary** number in the range ~~**0 … 32767**~~ **-131072 … +131071**
stringConstant: `"` a sequence of Unicode characters, not including double quote ~~**or newline**~~**`"`**
floatConstant: a single precision floating point number

identifier: a sequence of letters, digits, and underscore (`_`) not starting with a digit.

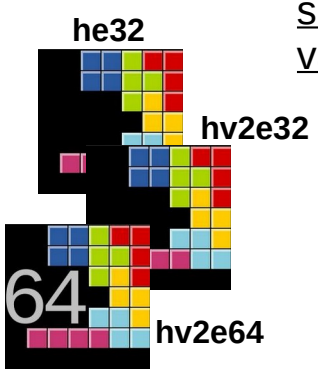


Language extensions

2. Program structure:

A Jack/XJack program is a collection of classes, each appearing in a separate file. A class is a sequence of tokens structured to the following context free syntax:

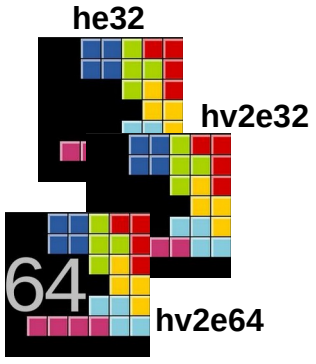
class:	`class` className `{` classVarDec* subroutineDec* `}`
classVarDec:	(`static` | `field`) type varName (`,` varName)* `;`
type:	`int` | `char` | `boolean` | **`float` |** className
subroutineDec:	**[`public` | `private`]**(`constructor` | `function` | `method`) (`void` | type) subroutineName `(` parameterList `)` subroutineBody
parameterList:	((type varName) (`,` type varName)*)?
subroutineBody:	`{` varDec* statements `}`
varDec:	`var` type varName (`,` type varName)* `;`
className:	identifier
subroutineName:	identifier
varName:	identifier



Language extensions

3. Statements:

statements:	statement*
statement:	letStatement | ifStatement | whileStatement | doStatement | returnStatement **| forStatement | switchStatement**
letStatement:	`let` varName (`[` expression `]`)? `=` expression `;`
ifStatement:	`if` `(` expression `)` `{` statements `}` (`else` `{` statements `}`)?
whileStatement:	`while` `(` expression `)` `{` (statement **| break | continue**)* `}`
doStatement:	`do` subroutineCall `;`
returnStatement:	`return` expression? `;`
forStatement:	**`for` `(` letStatement expression `;` letStatement `)` `{` (statement | break | continue)* `}`**
switchStatement:	**`switch` `(` intExpression `)` `{` (intConstant ,:‘ statements)* `default` ,:‘ statements `}`**

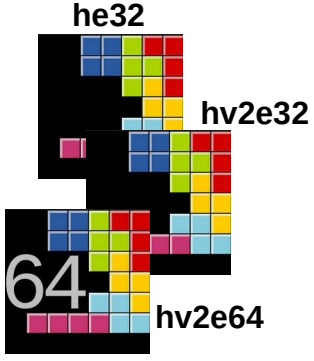


HV2

Language extensions

4. Expressions:

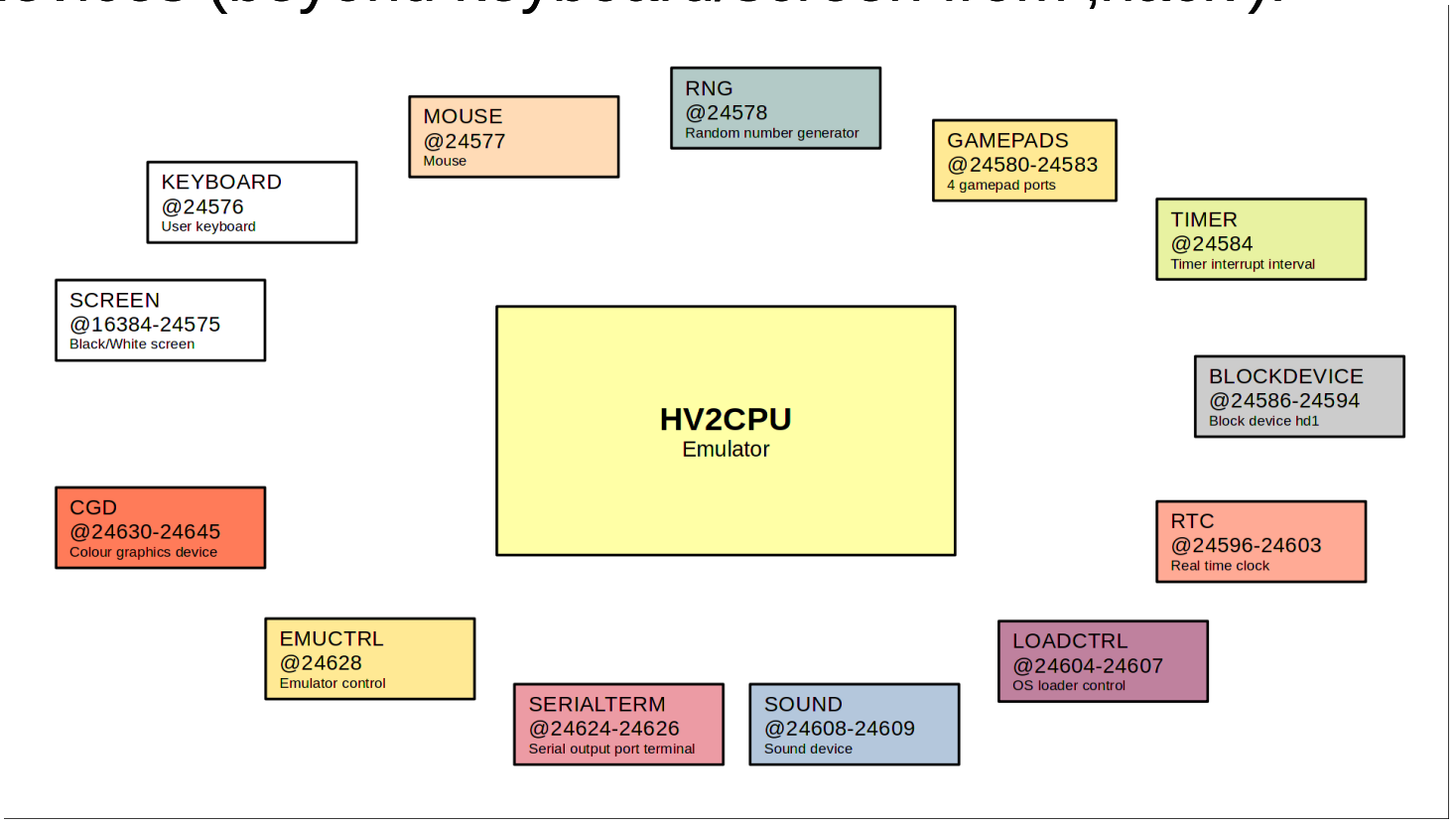
expression:	term (op term)*
term:	integerConstant | stringConstant | **floatConstant |** keywordConstant | varName (`[` expression `]`)? | subroutineCall | `(`expression `)` | unaryOp term
subroutineCall:	subroutineName `(` expressionList `)` | (className | varName) `.` subroutineName `(` expressionList `)` | **varName `(` expressionList `)`**
expressionList:	(expression (`,` expression)*)?
op:	`+` | `-` | `*` | `/` | `&` | `|` | `=` | `>` | `<`
unaryOp:	`-` | `~`
keywordConstant:	`true` | `false` | `null` | `this`

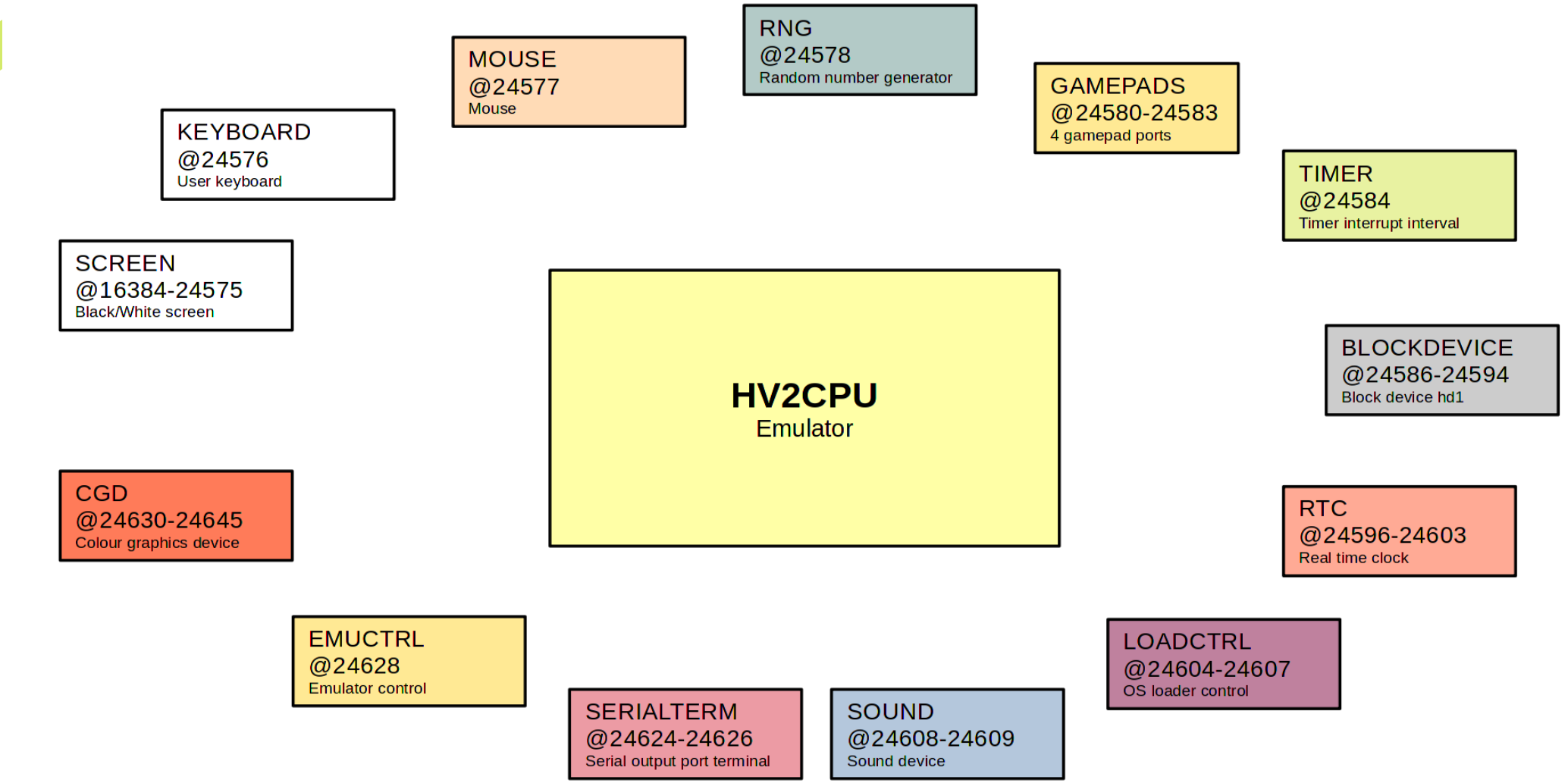


Other extensions:
No (empty) return statement in void functions needed. Function pointers permitted. Operator precedence! Entry point Main.main() is provided w/ argc, argv & envp ...

Device Support

The ‚Hack v2' platform provides memory mapped I/O devices (beyond keyboard/screen from ‚hack'):

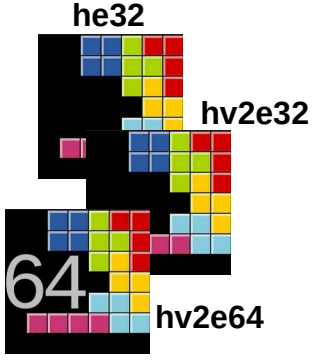



RNG
@24578
Random number generator
MOUSE
@24577
Mouse
GAMEPADS
@24580-24583
4 gamepad ports
KEYBOARD
@24576
User keyboard
TIMER
@24584
Timer interrupt interval
SCREEN
@16384-24575
Black/White screen
HV2CPU
Emulator
BLOCKDEVICE
@24586-24594
Block device hd1
CGD
@24630-24645
Colour graphics device
RTC
@24596-24603
Real time clock
EMUCTRL
@24628
Emulator control
LOADCTRL
@24604-24607
OS loader control
SERIALTERM
@24624-24626
Serial output port terminal
SOUND
@24608-24609
Sound device

Operating System

The HV2 platform operating system HV2OS:

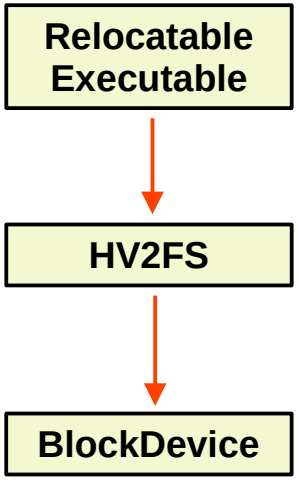
- will build on the original ‚Hack' (BI)OS.
- contains drivers for the additional devices
- supports platform specific features (timer interrupt etc.)
- has some higher level support (file system, task management, process management)
- shall NOT be linked to every application (like a framework or on MCs/SOCs)



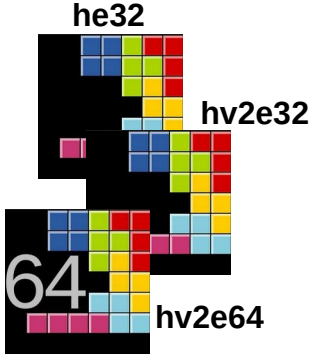
Operating System

On top of the file system*, an executable format is provided:

The executable will provide all necessary information for a successful relocation on the target system.



Offset	Len	Description
0	4	‘HV2R’, indicator ‘this is a Hack V2 relocatable binary’
4	4	Length of path to source directory (0 if w/o debug symbols)
8	<nlen>	Path to source directory (compiled w/ debug option only!)
<8+nlen>	4	No. of relocation entries to follow
<12+nlen>	8	Relocation entry #0
<20+nlen>	8	Relocation entry #1
…	...	...
<n>	8	Relocation entry #<n>
<n+8>	4	No. of static variable entries
<n+12>	4	Length of binary to follow
<n+16>	<len>	The actual binary



Each relocation entry consists of the following:

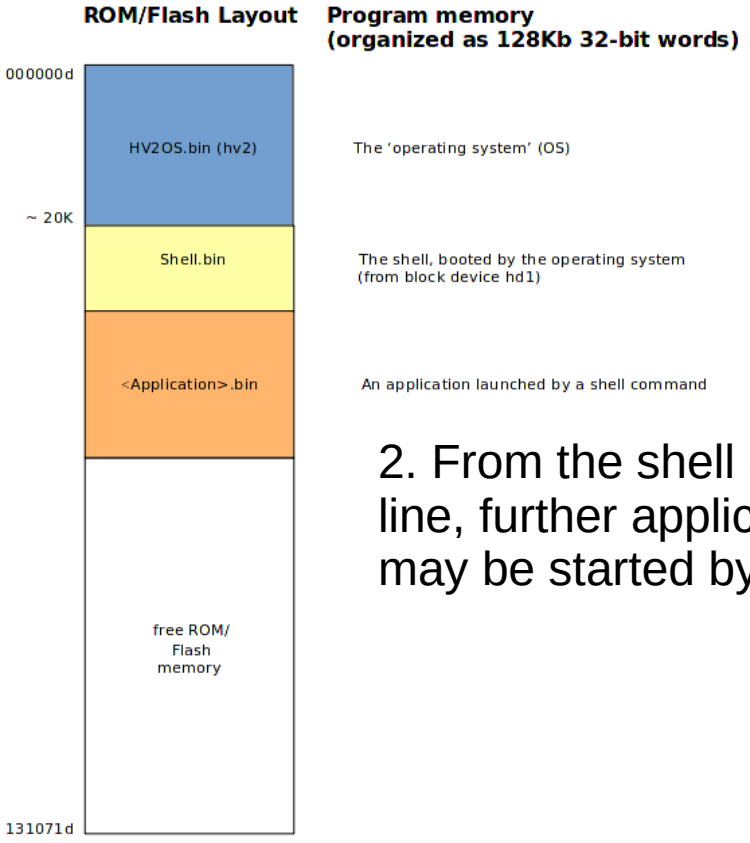
Offset	Len	Description
0	4	Offset to patch in the following binary
4	4	Current relative address (base 0) to be converted to the effective (relative) address

*see https://git.hacknology.de/kaqu/hv2e32/src/branch/master/doc/HV2FS.pdf for details

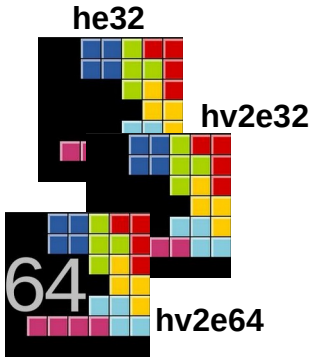
Operating System

HV2OS will be loaded upon boot into program memory (‚ROM/FLASH‘), starting at offset zero.

1. The OS initializes all its modules. From the ‚BlockDevice‘ it boots the 1st application, the ‚Shell.bin‘.



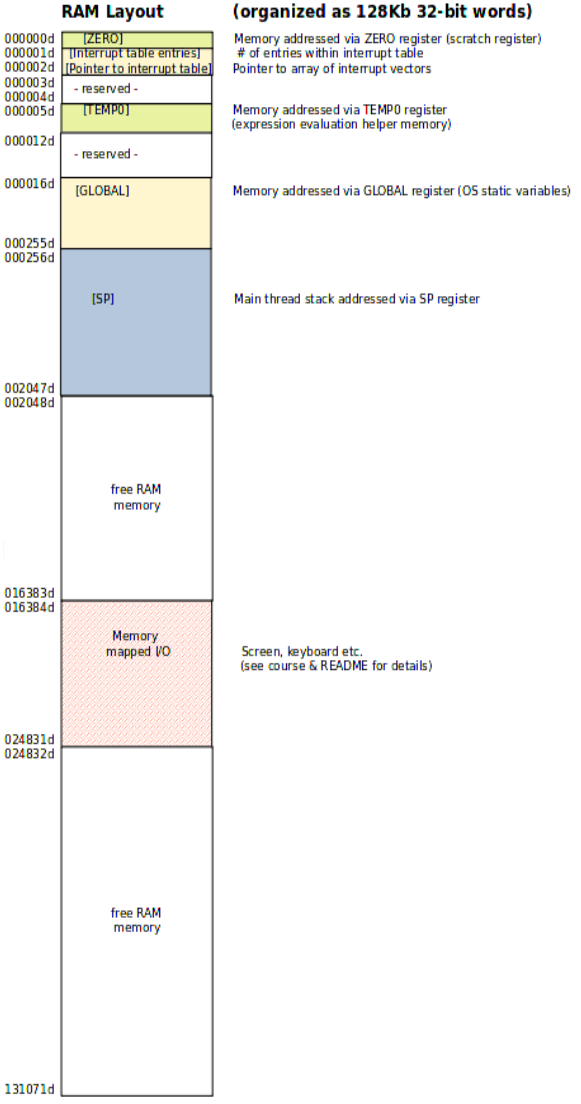
2. From the shell command line, further applications may be started by the user.

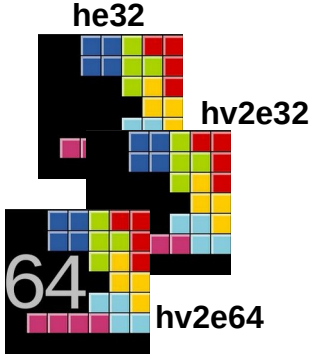


HV2

Operating System

The data memory (RAM)





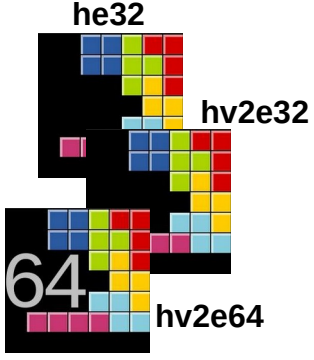
Operating System

HV2OS currently (2023) provides the following modules:

- Array
- **BlockDevice**
- **Emulator**
- **Environment**
- **FileSystem (& FileSystemEntry)**
- **HWTimer**
- **Interrupt**
- Keyboard **(enhanced)**
- Math **(enhanced)**
- Memory **(enhanced)**
- Output **(enhanced)**
- **Rand**
- **Relocator**
- **RTC**
- Screen **(enhanced)**
- **Semaphore**
- **Sound**
- String **(enhanced)**
- Sys **(enhanced)**
- **Tasks**
- **ColorGraphicsDevice**
- **Mouse**

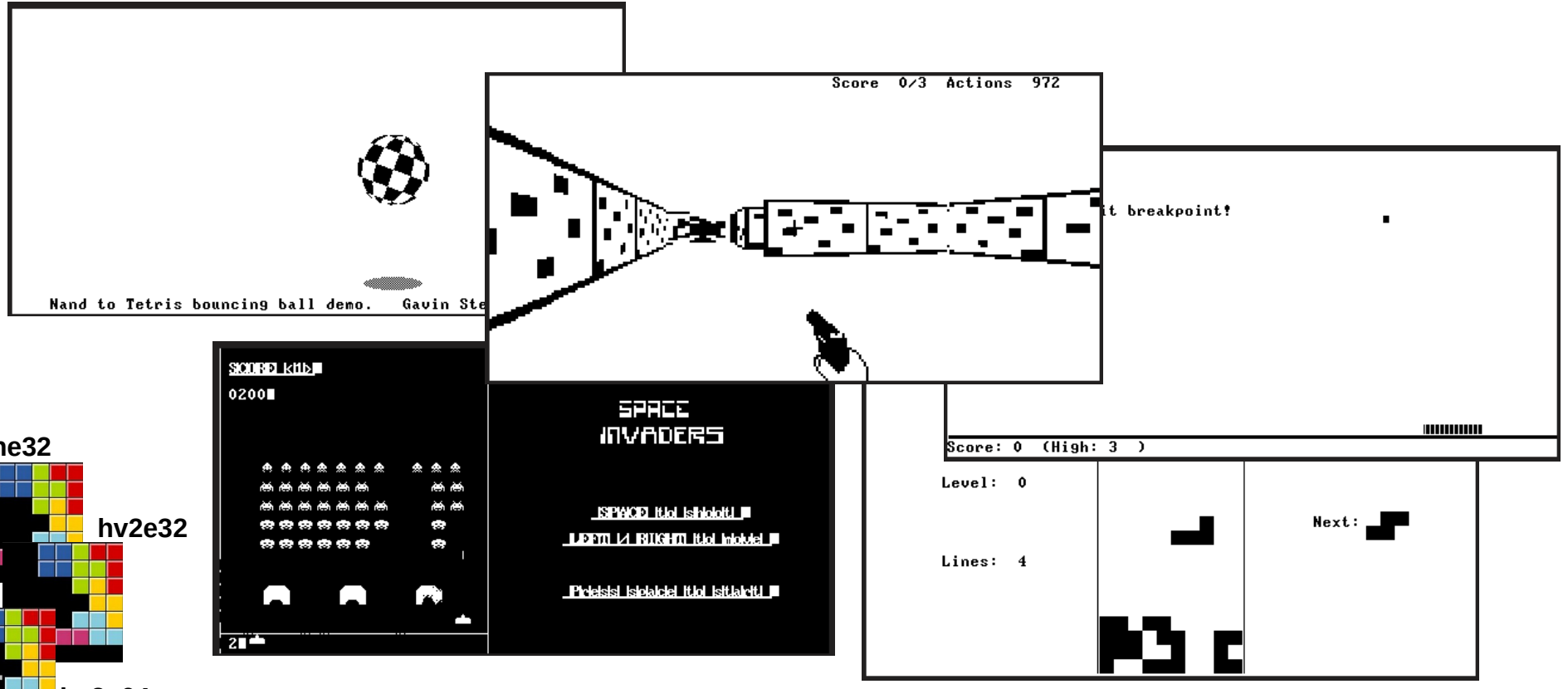
Applications make relative calls into the OS. Example:

```
...
private function void myTest() {
        var char c;

        let c = Keyboard.readChar();
        ...
}
...
```



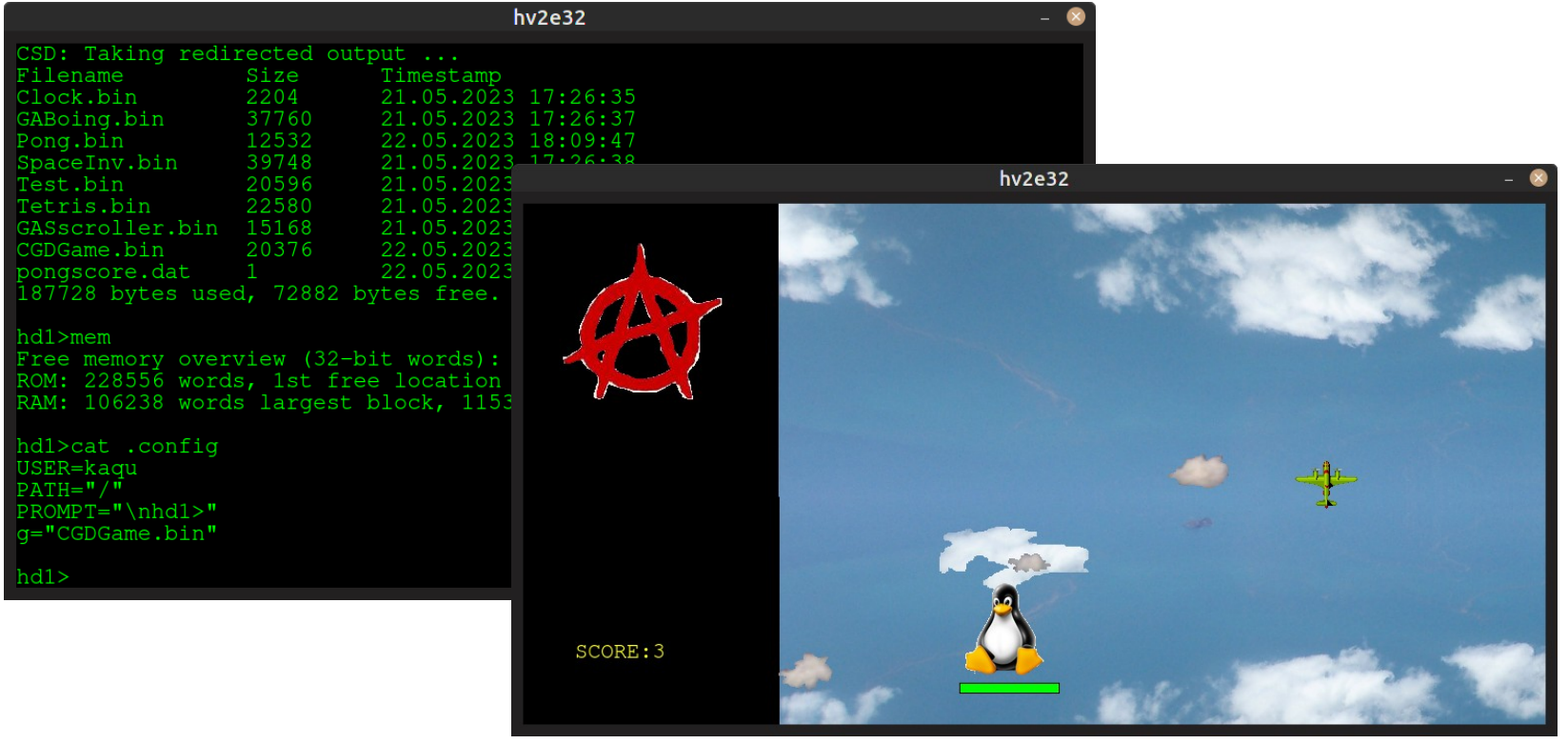
Operating System

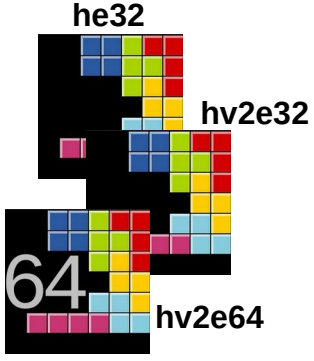
The black & white screen supports the original ‚hack' platform's games & demos.



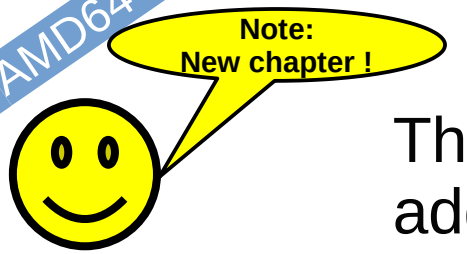
Operating System

The color graphics device (CGD) contains a sprite engine for easy game programming* (& there's sound too ...).



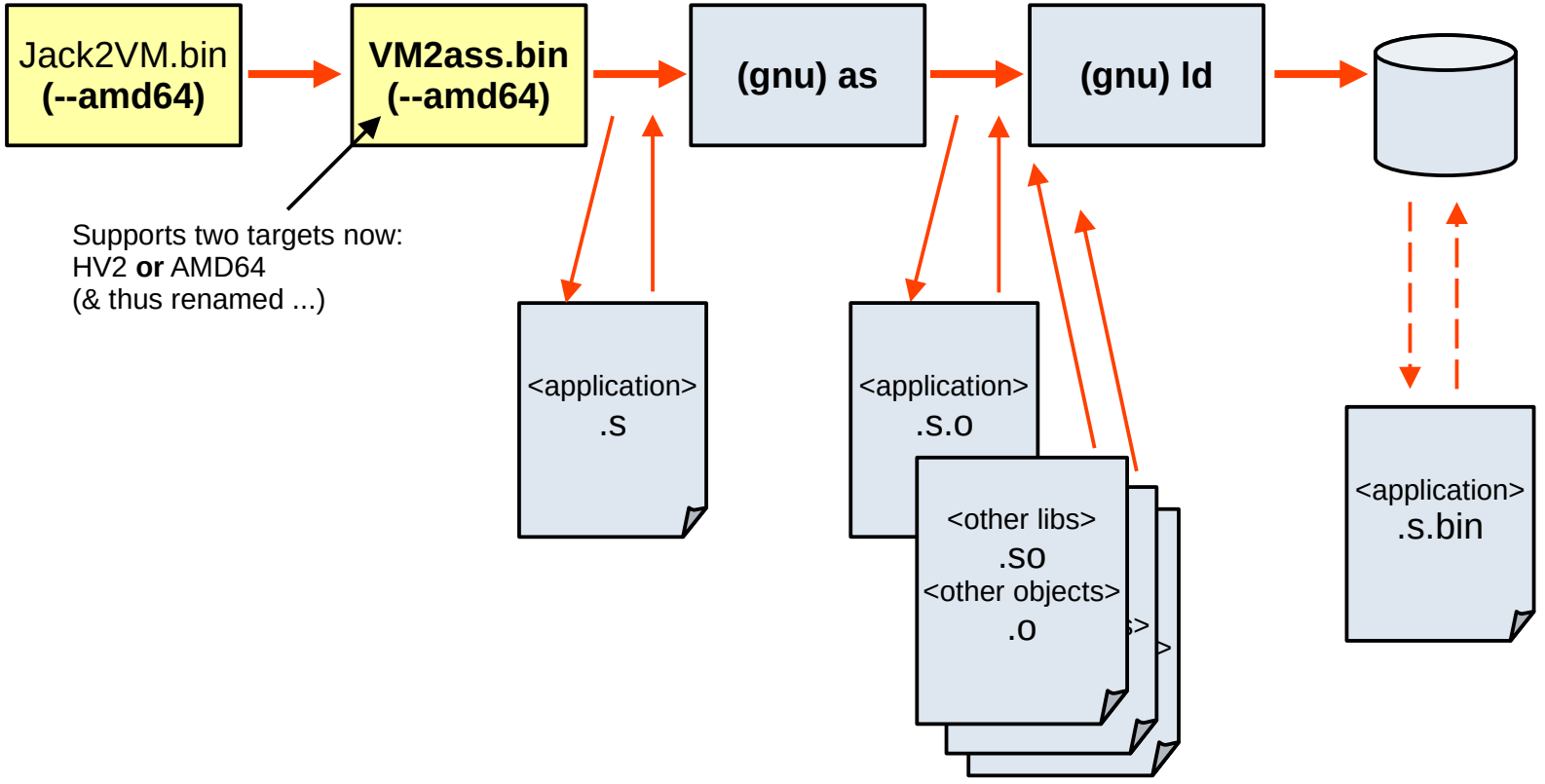


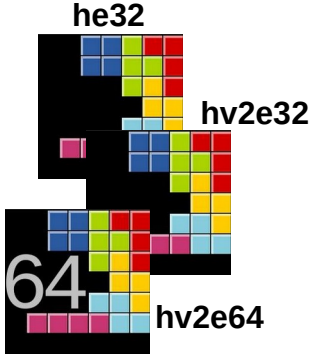
*see https://git.hacknology.de/kaqu/hv2e32/src/branch/master/doc/jack_on_HV2.pdf for details



Another target

The AMD64/x86_64 platform w/ SysV/Linux ABI has been added as a new ‚native‘ target.

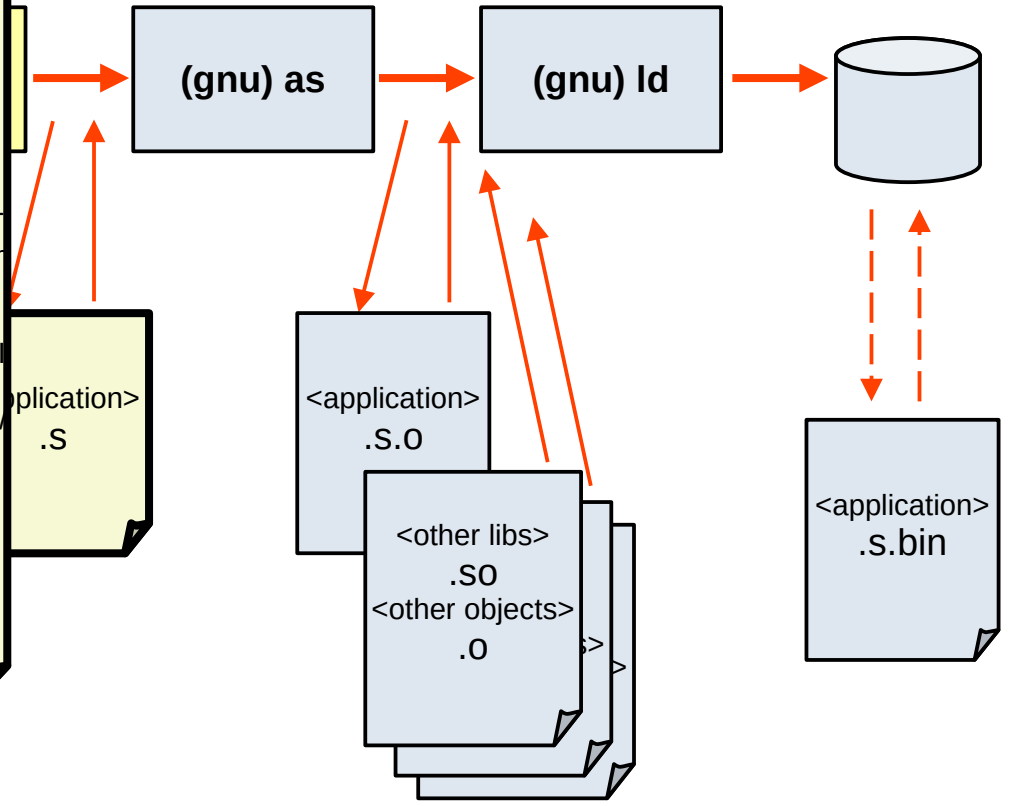




Another target

platform w/ SysV/Linux ABI has been
ve‘ target.

```
// L12085 - //@LINE#71#/media/kaqu/USR1/projects/HV2CPU/VM2ass
// L12086 - label Global.Main.terminaltest.LABEL_0_WHILE_START
// label: Main.Main.terminaltest$Global.Main.terminaltest.LABEL_0_WH
Main.Main.terminaltest$Global.Main.terminaltest.LABEL_0_WHILE_ST

// L12087 - push constant 0
// constant_push: 0
        pushq $0

// L12088 - not // true
// not: [TOS] = ![TOS]
        notq        (%rsp)          # [rsp] = ![rsp]

// L12089 - if-not-goto Global.Main.terminaltest.LABEL_1_WHILE_EXIT
// if-not-goto: if !<cond> goto Main.Main.terminaltest$Global.Main.termin
        popq        %rax            # rax = [rsp++]
        cmpq        $0, %rax            # rax == 0?
        je Main.Main.terminaltest$Global.Main.terminaltest.LABEL_1_WHI

// L12090 - //@LINE#72#/media/kaqu/USR1/projects/HV2CPU/VM2ass/
// L12091 - call Keyboard.keyPressed 0
// call Keyboard.keyPressed 0       # No of arguments: 0

        call Keyboard.keyPressed                # SP needs to be 16-Byte ali

        pushq %rax                          # HV2 ABI requires return value o

// L12092 - pop local 0  // c<=
// local_pop: RAM[LCL + #0] = TOS
        popq -56-8*0(%rbp)          # [BP - 56 - 8*0] = [%rsp++]
```



Another target

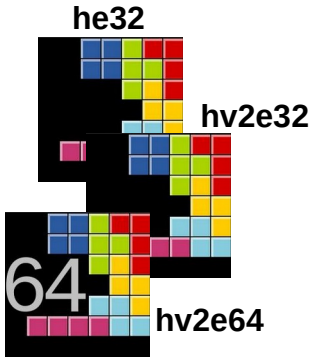
The AMD64 platform differs massively from the ‚Hack V2' platform:

HV2 features 16 integer registers, named R0 to R15 & 16 floating point ones (F0 to F15)
The assembler shall use these registers as follows:

Registerfile	Alternative name	points to / used as
R0|F0	SP|F0	Stack pointer
R1|F1	LCL|F1	Local variables base
R2|F2	ARG|F2	Arguments base
R3|F3	THIS|F3	Object base
R4|F4	THAT|F4	Array base
R5|F5	TEMP0|FTEMP0	->RAM[5]
R6|F6	GLOBAL|F6	Global static variables base (i.e. OS)
R7|F7	ZERO|FZERO	The 'empty' register (should have been R0 really ... ;)
R8|F8	STATIC|F8	Local static variables base (i.e. relocatables)
R9|F9	ARG1/TEMP7 | FTEMP7	Argument #1 (Compiler shall decide ARG/TEMP usage)
R10|F10	ARG2/TEMP6 | FTEMP6	Argument #2
R11|F11	ARG3/TEMP5 | FTEMP5	Argument #2
R12|F12	TEMP4/ARG4 | FTEMP4	Temporary values (4th assignment)
R13|F13	TEMP3/ARG5 | FTEMP3	Temporary values (3rd assignment)
R14|F14	TEMP2/ARG6 | FTEMP2	Temporary values (2nd assignment)
R15|F15	TEMP1/ARG7 | FTEMP1	Temporary values (1st assignment)

AMD64 features 16 integer registers & <n> floating point ones
For the amd64/x86_64 64-bit platform – matching closely the SysV/Linux ABI - the usage is as follows:

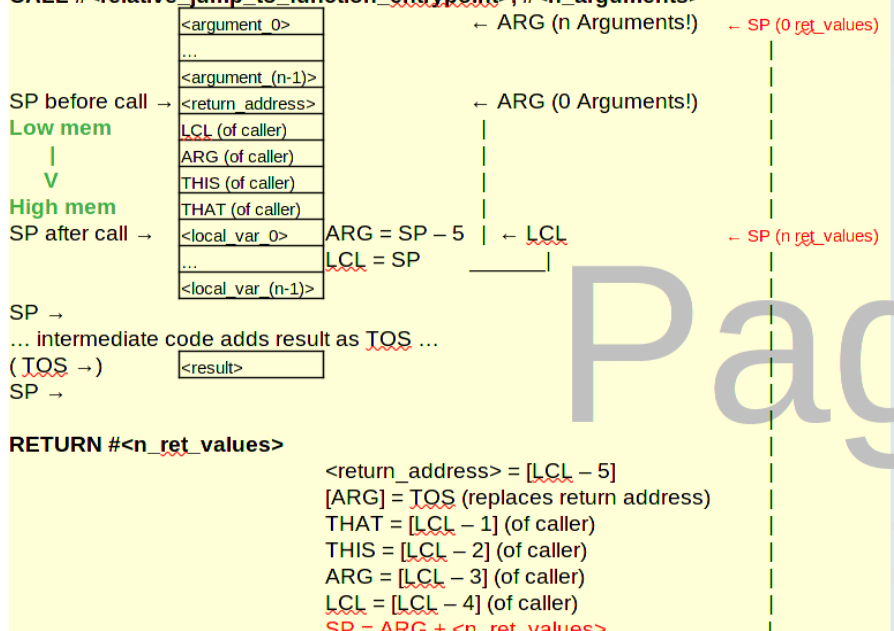
Registerfile w/o floats (Cs=Caller saved)	points to / used as
rsp	Stack pointer
rbp	Local variables & argument 0-5 base (base pointer)
	Arguments base: 48 + <n-6>*8 (if n > 6)
r12	Object base
r13	Array base
r15	TEMP0 based vars. (a pre-allocated memory area)
	- GLOBAL not used -
	- ZERO not used -
r14	Static variables (a pre-allocated memory area)
rdi (Cs)	Argument #0
rsi (Cs)	Argument #1
rdx (Cs)	Argument #2
rcx (Cs)	Argument #3
r8 (Cs)	Argument #4
r9 (Cs)	Argument #5
rax	Temporary #0, function return value
r10 (Cs)	Temporary #1
r11 (Cs)	Temporary #2
rbx	- reserved, saved by callee! -



Another target

HV2 ABI: Stack grows from lower to higher addresses (upwards)
The stack pointer SP points always above(!) TOS (TOS+1)
Function arguments are supplied on stack in the order of occurrence.

CALL #<relative_jump_to_function_entrypoint>, #<n_arguments>

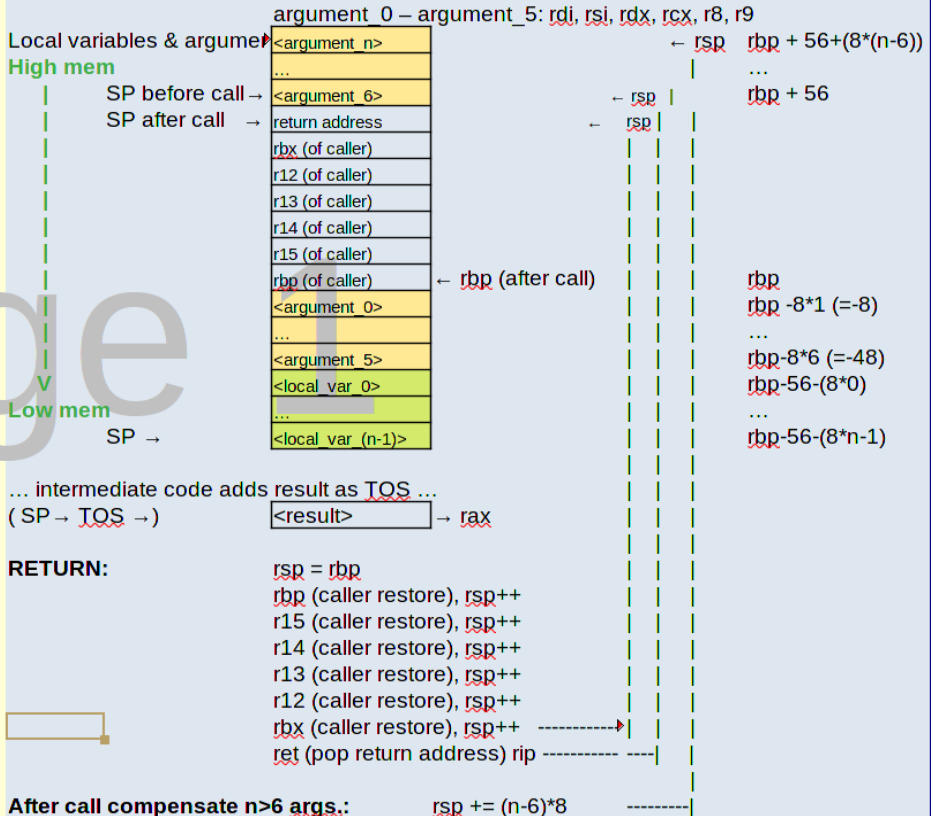


RETURN #<n_ret_values>

<return_address> = [LCL – 5]
[ARG] = TOS (replaces return address)
THAT = [LCL – 1] (of caller)
THIS = [LCL – 2] (of caller)
ARG = [LCL – 3] (of caller)
LCL = [LCL – 4] (of caller)
SP = ARG + <n_ret_values>
pc=<return_address>

AMD64/SysV ABI: Stack grows from higher to lower (downwards)
The stack pointer SP (rsp) points directly(!) to TOS
Function arguments on stack are supplied in reverse order, first six in registers.

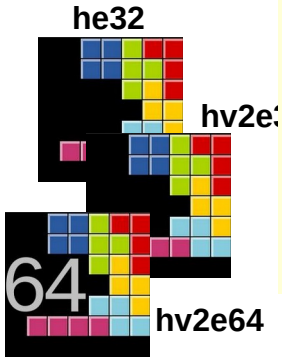
CALL <relative_jump_to_function_entrypoint>



RETURN:

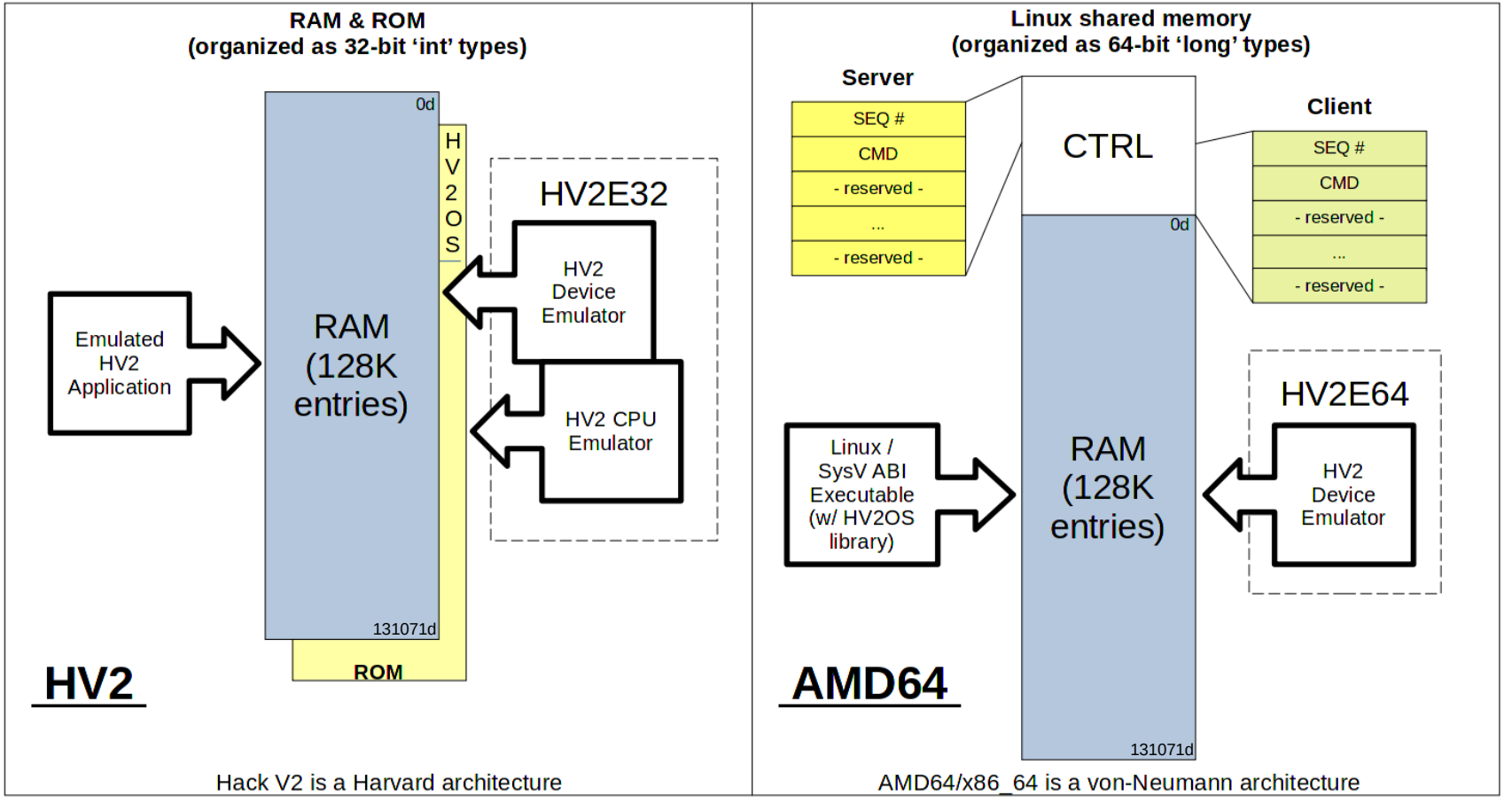
rsp = rbp
rbp (caller restore), rsp++
r15 (caller restore), rsp++
r14 (caller restore), rsp++
r13 (caller restore), rsp++
r12 (caller restore), rsp++
rbx (caller restore), rsp++
ret (pop return address) rip

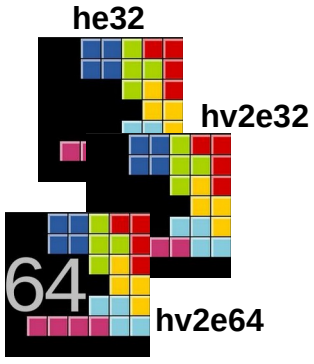
After call compensate n>6 args.: rsp += (n-6)*8
Return value: rax (needs to be pushed for Xjack toolchain!)



Another target

There is a HV2OS library available to enable XJack source compatibility. This requires shared memory to communicate with HV2 devices, provided by **hv2e64**.





Another target

‚Pre-processor' integration:

To permit the HV2OS libs to be used with different targets, it became necessary to permit for conditional compilation – mainly for inline assembly parts but also to slip in platform specific function call redirections.

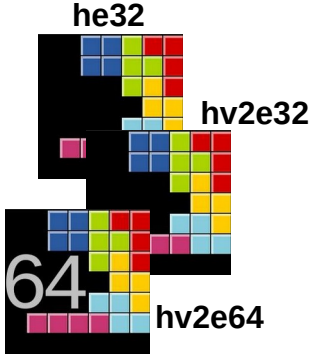
… common source code …

#ifdef HV2
    … some source code for the HV2 target platform only …
#endif

… common source code …

#ifdef AMD64
    … some source code for the AMD64 target platform only …
#endif

… common source code …



This is why there is an additional `- - amd64` parameter for Jack2VM.bin!

Another target

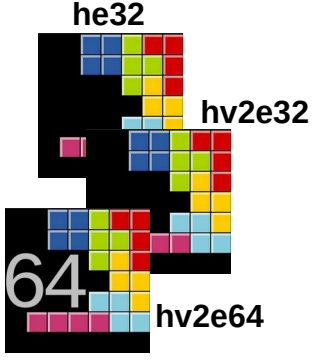
External library call support:

To permit calling of external libs, the ‚module' name ‚Extern' is special. So, instead of calling the external libc function directly, this prefix has to be applied:

```
public function int do_sleep_ms(int delay) { // A simple (XJack) wrapper
        // Now call the external function
        return Extern.usleep(delay * 1000); // µs!
}
```

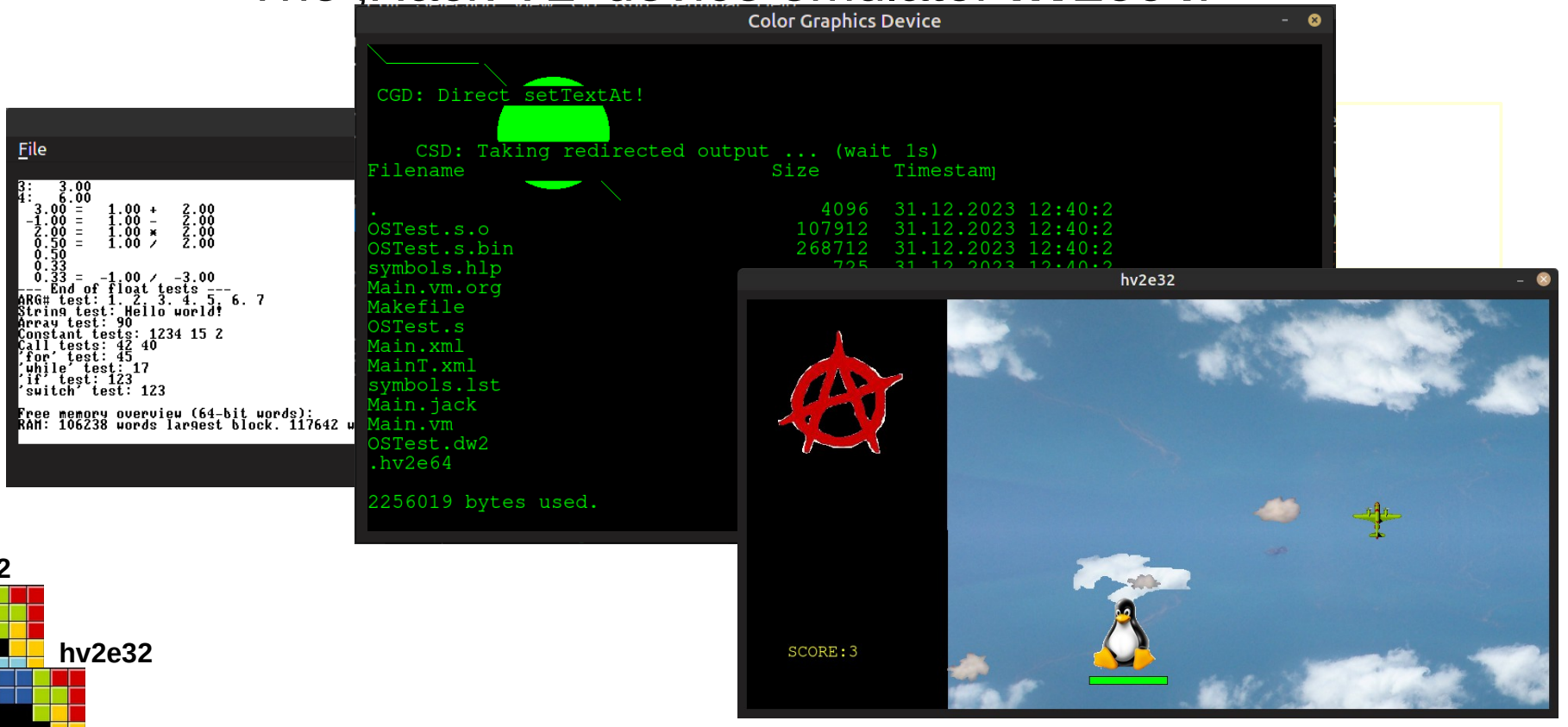
For this to work, an external file (os_symbols.lst) has to be provided w/ some hinting, concerning function signatures:

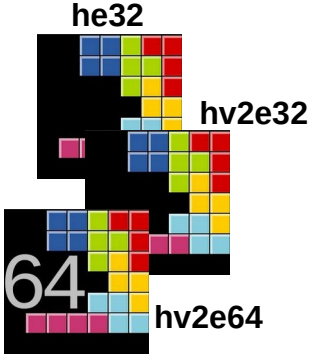
```
...
Extern.usleep function:3 int
Extern.usleep argument:0 int duration
...
```



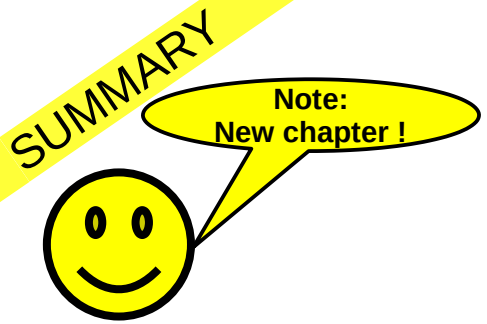
Another target

The ‚Hack V2' device emulator **hv2e64**:





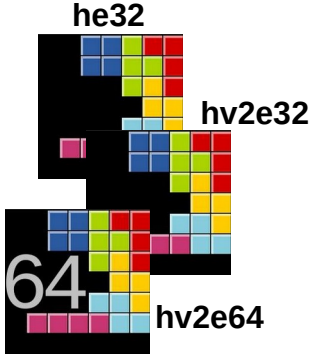
(see https://git.hacknology.de/kaqu/hv2e64 for details)



Summary

Land Ho!

(You almost made it!)



SUMMARY

Summary

Toolchain assembly output for comparison :

HACK

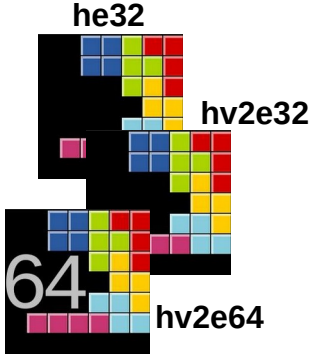
```
(Ball.bounce)    // Ball.bounce entry p
                 // using 5 local variables
@0
D=A
@SP
A=M
M=D     // [SP] = <value>

@SP
M=M+1   // ++SP

@0
D=A
@SP
A=M
M=D     // [SP] = <value>

@SP
M=M+1   // ++SP

@0
D=A
@SP
A=M
M=D     // [SP] = <value>
```

HV2

```
@LINE#223#jack/apps/Pong/Ball.jack#:
Ball.bounce:
L SP, SP + #5
PUSH [ARG + #0]
POP THIS
@LINE#227#jack/apps/Pong/Ball.jack#:
PUSH [THIS + #2]
PUSH #10
POP TEMP1
S [ZERO], TEMP1
IVTOS [ZERO]
EC SP
 [LCL + #2], [SP]
@LINE#228#jack/apps/Pong/Ball.jack#:
PUSH [THIS + #3]
PUSH #10
POP TEMP1
S [ZERO], TEMP1
DIVTOS [ZERO]
DEC SP
S [LCL + #3], [SP]
@LINE#229#jack/apps/Pong/Ball.jack#:
PUSH [ARG + #1]
PUSH #0
DEC SP
```

AMD64

```
// L12085 - //@LINE#71#/media/kaqu/US
// L12086 - label Global.Main.terminaltest
// label: Main.Main.terminaltest$Global.M
Main.Main.terminaltest$Global.Main.term

// L12087 - push constant 0
// constant_push: 0
        pushq $0

// L12088 - not // true
// not: [TOS] = ![TOS]
        notq        (%rsp)          # [rsp] = ![rs

// L12089 - if-not-goto Global.Main.termin
// if-not-goto: if !<cond> goto Main.Main.te
        popq        %rax            # rax = [rsp+
        cmpq        $0, %rax            # rax == 0
        je Main.Main.terminaltest$Global.Ma

// L12090 - //@LINE#72#/media/kaqu/US
// L12091 - call Keyboard.keyPressed 0
// call Keyboard.keyPressed 0       # No of

        call Keyboard.keyPressed

        pushq %rax
```

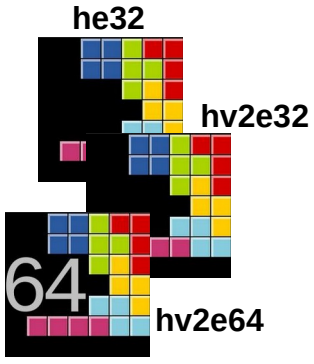


SUMMARY

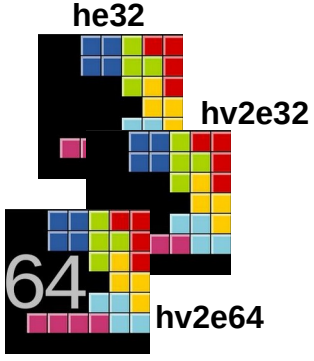
Summary

Code efficiency (example ‚Pong/Ball.bounce' 42 LOC):

	HACK	HV2	AMD64
Instructions	1903	176	230
Mem. Bytes	3806	704	738
Speed	Slow, just enough to be played ...	Too fast, needs braking!	Too fast, Needs braking



Thank you!



Watch the Red Hat guy dancing:
https://www.youtube.com/watch?v=SOJSM46nWwo