The Cm Virtual Machine Specification for Small-footprint Embedded Systems

Michel de Champlain
Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada

November 4, 2020

Table of Contents

Chapter 1

The Cm Virtual Machine Instruction Set

In this chapter, we discuss the instruction set of a virtual machine (VM) to support programming languages intended for small footprint embedded systems. The instruction set of the VM is tailored to support a subset of the C programming language, called Cm[1], intended for a restrictive microcontroller environment such as an ATmega368P 8-bit microcontroller with 32K bytes Flash and 2K bytes SRAM used in the Arduino Nano.

We carefully cover instruction formats, addressing modes and type representation as well as introduce the entire instruction set with practical examples.

[1] **Cm** as a subset of the **C** programming language for **m**icrocontrollers. In music, Cm or C- means C minor. A C minor chord is a chord that has C as a root :)

Before moving on, we present in Table 1.1, some naming conventions used to express the size and range of fields within operation codes and operands.

Symbol	Meaning and Size	Range Value
<i>	signed number	<i3> or <i4> or <i8> or <i16> or <i32>
<u>	unsigned number	<u3> or <u4> or <u8> or <u16> or <u32>
<v>	value	<i> or <u>
<n>	number	<i> or <u>
<a>	address	<u>
<o>	offset	<i>
<u3>	3-bit unsigned	0..7
<i3>	3-bit signed	-4..3
<u4>	4-bit unsigned nibble	0..15
<i4>	4-bit signed nibble	-8..7
<u5>	5-bit unsigned	0..31
<i5>	5-bit signed	-16..15
<u8>	8-bit unsigned	0..255
<i8>	8-bit signed	-128..127
<u16>	16-bit unsigned	0..65535
<i16>	16-bit signed	-32768..32767
<u32>	32-bit unsigned	0..4294967295
<i32>	32-bit signed	-2147483648..2147483647

Table 1.1: Instruction Format Naming Conventions.

1.1 Instruction Formats

Instruction formats determine the layout and size for each instruction of a virtual machine. Not surprisingly, the choice of instruction format is a fundamental design decision and involves several factors. instruction formats

The first factor to consider is the instruction size itself. Making instructions short is especially important for embedded systems where memory is a limited resource. But keeping the size of an instruction very small can make it harder to decode in order to execute it. In general though, an instruction consists of an **operation code** (opcode) immediately followed by operands (or instruction parameters).

The **Cm VM** instruction formats are quite straightforward and come in one of three main formats. The **inherent** format has no operands and is self-contained in one byte, including immediate operands and displacements. The **byte-parameter** format has a single one-byte operand and requires two bytes of memory. And the **word-parameter** format has a single two-byte operand and requires three bytes of memory. All opcodes and most instructions of **Cm VM** are in inherent format and therefore require only a single byte. Many instructions, too, result in data transfer to and from the operand (32-bit) stack.

All formats are shown in Figure 1.1 below.

```
Format:
          +---------+
Inherent  |  opcode |
          +---------+

byte or   +---------+---------+
8-bit     |  opcode | operand |  operand = <i8> or <u8>
Operand   +---------+---------+

          +---------+-------------------+
16-bit    |  opcode |      operand      |  operand = <i16> or <u16>
Operand   +---------+-------------------+

          +---------+---------------------------------------+
32-bit    |  opcode |                operand                | operand = <i32> or <u32>
Operand   +---------+---------------------------------------+

Bits      |<---8--->|<---8--->|<---8--->|<---8--->|<---8--->|
```

Figure 1.1: Instruction Formats for **Cm VM**.

A second factor to consider ensures that there is sufficient space in the instruction format to express all operations required.

The third factor to consider is the number of bits in an address field. In our case, making the 8-bit byte as the basic unit of memory was the most realistic option for 8-bit microcontrollers. The maximum addressable memory of the **Cm VM** is 64K.

The fourth factor is concerned with the usage of relative addresses. Relative addressing allows position-independent code meaning that the virtual machine code can be loaded anywhere in memory. The generation of position-independent code follows one important rule of never using absolute addressing. This is achieved by using the instruction pointer (`ip`) as the base register for a relative offset. **Cm VM** mainly uses relative offset for flow control (branching and calling). Very short branches are optimized by embedding an immediate 5-bit offset in a one byte opcode where the range is limited to +15 or -16 bytes from the following opcode. For short branches, the byte following the branch opcode is treated as an 8-bit offset to be used to calculate the effective address of the next instruction. Finally, long branches require 16-bit offsets. Because instructions are three bytes, long branches are expensive in terms of space.

1.2 Addressing Modes

Addressing modes specify where operands are to be retrieved, either from memory, registers, accumulators, stacks and so on. Bearing in mind the tiny nature of our embedded systems, two general methods may be used to reduce the addressing size of operands within instructions: addressing modes

- Move the operand into a register when it is used several times.
- Use a single specification to select operands.

The above methods work well for simple operations, but are a nightmare when several intermediate results are needed. By exploiting the stack machine architecture and using the operand stack for our instruction set, we can eliminate a number of non-applicable addressing modes such as direct, register, register indirect, and so on. Consequently, only the four addressing modes below are efficiently supported by **Cm VM**:

1. Stack (or inherent),
2. Immediate, and
3. Relative.

For **stack** or **inherent addressing**, otherwise known as zero-address instructions, both source and destination operands are implicitly retrieved from the operand stack. This makes virtual machine instructions as short as possible by reducing address lengths to zero. Hence, inherent instructions have no operands and are self-contained in a single byte. Table 1.2 illustrates all inherent instructions sorted by opcode.

inherent format

Hex	**Binary**	**Mnemonic**	**Operand**	**Description**	**Operation**
00	000 00000	halt		Stop virtual machine	
01	000 00001	pop		Remove top of stack	[... = v
02	000 00010	dup		Duplicate top of stack	[r r = v
03	000 00011	exit		Return from function with parameters	
04	000 00100	ret		Return from function	
05	000 00101	—		Reserved for future used	
06	000 00110	—		Reserved for future used	
07	000 00111	—		Reserved for future used	
08	000 01000	—		Reserved for future used	
09	000 01001	—		Reserved for future used	
0A	000 01010	—		Reserved for future used	
0B	000 01011	—		Reserved for future used	
0C	000 01100	not		Bitwise one's complement	[r = ~v
0D	000 01101	and		Bitwise AND	[r = v1 & v2
0E	000 01110	or		Bitwise OR	[r = v1 | v2
0F	000 01111	xor		Bitwise exclusive OR	[r = v1 ^ v2
10	000 10000	neg		Negate	[r = -v
11	000 10001	inc		Increment	[r = ++v
12	000 10010	dec		Decrement	[r = --v
13	000 10011	add		Addition	[r = v1 + v2
14	000 10100	sub		Subtraction	[r = v1 - v2
15	000 10101	mul		Multiplication	[r = v1 * v2
16	000 10110	div		Division	[r = v1 / v2
17	000 10111	rem		Remainder, modulo	[r = v1 % v2
18	000 11000	shl		Shift left	[r = v1 << v2
19	000 11001	shr		Shift right	[r = v1 >> v2
1A	000 11010	teq		Test for equal	[r = v1 == v2
1B	000 11011	tne		Test for not equal	[r = v1 != v2
1C	000 11100	tlt		Test for less than	[r = v1 < v2
1D	000 11101	tgt		Test for greater than	[r = v1 > v2
1E	000 11110	tle		Test for less or equal	[r = v1 <= v2
1F	000 11111	tge		Test for greater or equal	[r = v1 >= v2

Table 1.2: Inherent (one byte, no operand) Instructions.

immediate format

For **immediate addressing**, the operand is included as part of the opcode itself and is automatically fetched in one byte. Hence, immediate instructions are also self-contained in a single byte. Although 8 bits is obviously limited, it is handy for specifying small integer literals. Within the immediate addressing mode, the format specifies one or more additional fields with different ranges (`<i3>`, `<u3>`, or `<i5>`) and subdivides this mode into further instruction groupings as shown in Table 1.3.

Hex	Mnemonic	Operand	Description	Operation
30..4F	`br.i5`	Label ($<i5>$)	Branch always	`pc += <i5>`
50..6F	`brf.15`	Label ($<i5>$)	Branch if v != 1	`if (TOS != 1) pc += <i5>`
70..8F	`enter.u5`	FctInfo ($<i5>$)	Set up frame	`See instruction section`
90..97	`ldc.i3`	$<i3>$	Load constant	`r = <i3>`
98..9F	`addv.u3`	$<u3>$	Add TOS to variable	`bp[<u3>] += TOS`
A0..A7	`ldv.u3`	$<u3>$	Load variable	`r = bp[<u3>]`
A8..AF	`stv.u3`	$<u3>$	Store variable	`bp[<u3>] = r`

Table 1.3: Immediate (one byte) Instructions.

relative format

Finally, for **relative addressing**, the opcode is followed by either a one byte (8-bit) or two byte (16-bit) operand. Immediate addressing, in this sense, is the optimized version of relative addressing. The operand represents an offset (`<i8>`, `<u8>`, or `<u16>`) or index (`<i8>`, `<u8>`, or `<u16>`), and is used to reference local variables and arguments. Within the relative addressing mode, the format also specifies fields of different ranges (`<i8>` or `<i16>`), and subdivides this mode into further instruction groupings as shown in Table 1.4.

Hex	Mnemonic	Operand	Description	Operation
B0	`addv.u8`	$<u8>$	Add TOS to variable	`bp[<u8>] += TOS`
B1	`ldv.u8`	$<u8>$	Load variable	`r = bp[<u8>]`
B2	`stv.u8`	$<u8>$	Store variable	`bp[<u8>] = r`
B3	`incv.u8`	$<u8>$	Increment variable	`++bp[<u8>]`
B4	`decv.u8`	$<u8>$	Decrement variable	`--bp[<u8>]`
BF	`enter.u8`	$<u8>$	Set up frame on function entry	`See instruction section`
D5	`lda.i16`	$<i16>$	Load address	`See instruction section`
D9	`ldc.i8`	$<i8>$	Load an 8-bit constant	`[r = <i8>`
DA	`ldc.i16`	$<i16>$	Load a 16-bit constant	`[r = <i16>`
DB	`ldc.i32`	$<i32>$	Load a 32-bit constant	`[r = <i32>`
E0	`br.i8`	Label ($<i8>$)	Branch relative always	`pc += <i8>`
E1	`br.i16`	Label ($<i16>$)	Branch relative always	`pc += <i16>`
E3	`brf.i8`	Label ($<i8>$)	Branch relative if false	`if (!r) pc += <i8>`
E7	`call.i16`	Label ($<i16>$)	Call relative	
FF	`trap`	$<u8>$	Trap to vector	`pc = vt[<u8>]`

Table 1.4: Relative (two, three, or four byte) Instructions.

1.3 Instruction Set

The following section provides an alphabetized listing of the entire **Cm VM** instruction set. A detailed description of each instruction makes up the bulk of this section (and chapter), and serves as a reference. Descriptions are presented in alphabetical order using the following format:

- The assembler syntax.
- A concise description of how it works.
- An ANSI C description of its corresponding operation. The description is designed for readability and not optimization. On the other hand, the target **Cm VM** also written in ANSI C, is optimized for maximum performance.
- The layout of the stack before the operation.
- The layout of the stack after the operation.
- One or more examples.

add

Addition

Assembler Syntax: `add`

Description: The `add` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1` + `v2` and is pushed back onto the operand stack.

Operation:

```
void add() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 + v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
                ; [...
ldc     2
                ; [2, ...
ldc     -3
                ; [2, -3, ...
add
                ; [-1, ...
```

addv Add Value to a Local Variable

Assembler Syntax: `addv <u3>`

Description: Adds a value to the content of the specified object local variable. The `addv` pops the `value` from the operand stack. A function parameter is also considered as a local variable (see the ordering and layout on the operand stack in the `enter`/`ret` instructions). This instruction has one operand `<u3>` which indicates the local variable number (offset in the current frame pointer `fp`) in the object specified to add.

Operation:

```
void addv(u3 localVarNumber) {
    i32 value = pop();

    fp[localVarNumber] += value;
}
```

Stack Before: `[value, ...`

Stack After: `[...`

Example:

```
public void fct() {
    int n;          // local variable 0

    // ...

    n += 3;

          ldc     3
                        ; [3, ...
          addv    0
                        ; [...
}
```

and

Bitwise And

Assembler Syntax: `and`

Description: The `and` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 & v2` and is pushed back onto the operand stack.

Operation:

```
void and() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 & v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
                ; [...
      ldc    0b0101
                ; [5, ...
      ldc    0b0110
                ; [5, 4, ...
      and
                ; [4, ...
```

br Branch at Address

Assembler Syntax: `br`

Description: Unconditional branch to relative or absolute address. The `br` adds the `offset` (if relative) or sets the `addr` (if absolute) to the instruction pointer `ip`.

Operation:

```
void brI8(i8 offset) { ip += offset; } // relative offset

void brU16(u16 addr) { ip  = addr; } // absolute address
```

Stack Before: `[...`

Stack After: `[...`

Example:

```
While

    ; ...

    br      While
```

brf

Branch If False at Address

Assembler Syntax: `brf`

Description: Conditional branch if the top of the operand stack is false. The `brf` pops the value `v` from the operand stack and adds the `offset` (if relative) or sets the `addr` (if absolute) to the instruction pointer `ip` if `v` is false. Otherwise, if `v` is true then one (if relative) or two (if absolute) is added to `ip`.

Operation:

```
void brfI8(i8 offset) { // relative offset
    bool v = (bool)pop();

    ip += v ? 1 : offset;
}

void brfU16(u16 addr) { // absolute address
    bool v = (bool)pop();

    ip  = v ? ip+2 : addr;
}
```

Stack Before: `[r, ...`

Stack After: `[...`

Example:

```
             ; [...
     ldc  3
             ; [3, ...
     ldc  2
             ; [3, 2, ...

If   tlt     ; if ( 3 < 2 )

             ; [0, ...
     brf  Else

; ...

Else
```

call — Call Function at Address

Assembler Syntax: `call <u8> or <u16>`

Description: The `call` pushes the return address `ra` onto the operand stack. The `call` with an `<i8>` operand adds the relative `offset` to the instruction pointer `ip`. The `call` with an `<u16>` operand replaces the the instruction pointer `ip` by the absolute address `<u16>`.

Operation:

```
void callI8(i8 offset) { // using a relative offset
    push(ip+1);
    ip += offset;
}

void callU16(u16 addr) { // to absolute address
    push(ip+2);
    ip = addr;
}
```

Stack Before: `[ra, ...`

Stack After: `[...`

Example:

```
    call    Fct
RA  ldc     0      ; label RA corresponds to the return address pushed

    ; ...

Fct
```

dec Decrement

Assembler Syntax: `dec`

Description: Decrements the top of the operand stack.

Operation:

```
void dec() {
    --stack[sp];
}
```

Stack Before: `[v, ...`

Stack After: `[v, ...`

Example:

```
                ; [...
ldc     10
                ; [10, ...
dec
                ; [9, ...
```

decv Decrement Variable

Assembler Syntax: `decv <u3>`

Description: Decrements the content of the specified object local variable. A function parameter is also considered as a local variable (see the ordering and layout on the operand stack in the `enter`/`ret` instructions). This instruction has one operand `<u3>` which indicates the local variable number in the object specified to decrement.

Operation:

```
void decv(u3 localVarNumber) {
    --stack[localVarNumber];
}
```

Stack Before: `[...`

Stack After: `[...`

Example:

```
public void fct() {
    int n;          // local variable 0

    // ...

    --n;

          addv    0
                        ; [...
}
```

div Divide

Assembler Syntax: `div`

Description: The `div` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 / v2` and is pushed back onto the operand stack.

Operation:

```
void div() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 / v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
                ; [...
    ldc     3
                ; [3, ...
    ldc     2
                ; [3, 2, ...
    div
                ; [1, ...
```

dup

Duplicate

Assembler Syntax: `dup`

Description: Duplicates the top item on the operand stack.

Operation:

```
void dup() {
    i32 v = pop();

    stack[++sp] = (i32)v;
    stack[++sp] = (i32)v;
}
```

Stack Before: `[v, ...`

Stack After: `[v, v, ...`

Example:

```
                ; [...
ldc     3
                ; [3, ...
dup
                ; [3, 3, ...
```

enter Set up Frame on Function Entry

Assembler Syntax: `enter <u5>` or `<u8>`

Description: The `enter` instruction is the first instruction of a function. It saves the frame context of its caller, and sets up the context of the current function. The `enter <u5>` instruction takes only one byte and has an immediate operand `u5` in the opcode. On the other hand, the `enter <u8>` instruction has a one-byte operand `u8`. Each operand represents important information about the frame context of the current function. This information is used by the instruction `ret` to clean up the operand stack. As such, the operand is divided into three fields containing a flag `v` if the function returns a value or not (`void`), the number of parameter(s) passed to a function (`np`), and the number of local variables to be allocated within the function (`nl`).

The instruction `enter <u5>` is optimized for functions up to a maximum of 3 parameters and 3 local variables. The instruction `enter <u8>` takes two bytes but permits up to 7 parameters and 7 local variables. To access local variables (including parameters) on the operand stack via the related instructions (`ldv`, `stv`, and so on), the following function is used:

```
int getFrameOffset(int v, int np, int nl) { return 2 + np + nl + v; }
```

Operation:

```
 7 6 5 4 3 2 1 0                        7 6 5 4 3 2 1 0
+-----+-+---+---+                      +-+-+-----+-----+
|0 1 1|v|np |nl |                      |x|v| np  | nl  |
+-----+-+---+---+                      +-+-+-----+-----+
   opcode.<u5>              opcode        <u8>

   (0x60..0x7F)              0xF5         <u8>



 7 6 5 4 3 2 1 0
+-+-+-----+-----+
|x|v| np  | nl  |
+-+-+-----+-----+
function info (fi)
```

```
void enter(int u5) {
    int   fi.v  = (u5 >> 4) & 0x01;
    int   fi.np = (u5 >> 2) & 0x03;
    int   fi.nl =  u5       & 0x03;
```

```
//  int   fi = (v << 6) | (np << 3) | nl;

    retAddr = stack[sp--];  // pop (save) caller's return address
    sp += nl;               // allocate space for local variables
    stack[++sp] = fi;       // push function info
    stack[++sp] = bp;       // push (save) caller's bp (context)
    bp = sp;                // set frame context for the current function
    stack[++sp] = retAddr;  // push back the caller's return address
}

void enter(int u8) {
    int   np = (u8 >> 4) & 0x0F;
    int   nl =  u8       & 0x0F;

    stack[++sp] = bp;  // push (save) caller's bp (context)
    bp = sp;           // set frame context for the current function
    sp += nl;          // allocate space for local variables
}
```

Stack Before:

```
sp -> | retAddr |
      |   pn-1  |
      ~   ...   ~
      |   p1    |
      |   p0    |
      +---------+

      [p0, p1, ..., pn-1, retAddr, ...
```

Stack After:

```
int getFrameOffset(int v, int np, int nl) { return 2 + np + nl + v; }


sp->bp->|caller bp| bp + 0
        | retAddr | bp - 1
        | fctInfo | bp - 2
        |   ln-1  | bp - 3
        ~   ...   ~   ...
        |   l1    |
        |   l0    |
        |   pn-1  |
        ~   ...   ~   ...
        |   p1    |
        |   p0    |
        +---------+

        [p0, p1, ..., pn-1, l0, l1, ..., ln-1, <u5>, ra, bp, ...
```

Example: The following is the stack state after the execution of the enter (∗):

```
bp -> |caller bp| bp + 0
      | retAddr | bp - 1
      |   <u5>  | bp - 2
      |   j     | bp - 3
      |   i     | bp - 4
      |   p     | bp - 5
      +---------+

      [p, retAddr, bp, i, j, ...

void fct(int p) {     // function with one parameter and two local variables
    int i, j;         // where v = 0, np = 0x01, and nl = 0x02

                      // enter  6  ; (0x01 << 2)|0x02
                      // (*)
                      //          ;                                bp - getFrameOffset(v,np,nl)
    j = i = p;        // ldv    0 ; load from stack[bp-2] or stack[bp - getFrameOffset(0, 1, 2)]
                      // dup
                      // stv    1 ; store to  stack[bp+1] or stack[bp - getFrameOffset(1, 1, 2)]
                      // stv    2 ; store to  stack[bp+2] or stack[bp - getFrameOffset(2, 1, 2)]
    //...
}
```

Note: `enter 0` is useless since it means to set up a frame with parameters and no local variables. In such a case, the instruction can be removed for optimization purposes. Hence, the Cm compiler is removes all `enter 0` when it generates the code.

halt

Stop Virtual Machine

Assembler Syntax: `halt`

Description: Stops the virtual machine. This instruction is also used to set breakpoints in the **CDotM**.

Operation:

```
void halt() {
    // Stop the virtual machine.
}
```

Stack Before: `[...`

Stack After: `[...`

Example:

```
                ; [...
ldc     3
                ; [3, ...
halt
```

inc Increment

Assembler Syntax: `inc`

Description: Increments the top of the operand stack.

Operation:

```
void inc() {
    ++stack[sp];
}
```

Stack Before: `[...`

Stack After: `[...`

Example:

```
                ; [...
ldc     2
                ; [2, ...
inc
                ; [3, ...
```

incv Increment Variable

Assembler Syntax: `incv <u3>`

Description: Increments the content of the specified object local variable. A function parameter is also considered as a local variable (see the ordering and layout on the operand stack in the `enter`/`ret` instructions). This instruction has one operand `<u3>` which indicates the local variable number in the function specified to increment.

Operation:

```
void incv(u3 localVarNumber) {
    ++stack[localVarNumber];
}
```

Stack Before: `[...`

Stack After: `[...`

Examples:

```
public void fct() {
    int n;          // local variable 0

    // ...

    ++n;

          incv    0
                         ; [...
}
```

```
module Counter {
    public int count;
    public void fct(ref Counter this, int p, Counter c) {
            enter  ?? ; offsets ==> this = 0; p = 1; c = 2; v = 3
        int v;

        v++;
            ldv    0  ; push this
            incv   3  ; this.v++

        p++;
            ldv    0  ; push this
            incv   1  ; this.p++

        c.count++;
            ldv    2  ; push this
            incf   0  ; this.count++

            ret
    }
}
```

ldc Load Constant

Assembler Syntax: `ldc <i3> or <i8> or <i16>`

Description: Loads a constant onto the operand stack. The `ldc` pushes the integer `<i>` onto the operand stack.

Operation:

```
void ldc(I3  i3)  { stack[++sp] = i3;  }  //     [-4..3]
void ldc(I8  i8)  { stack[++sp] = i8;  }  //   [-128..127]
void ldc(I16 i16) { stack[++sp] = i16; }  // [-32768..32767]
```

Stack Before: `[...`

Stack After: `[<i>, ...`

Where `<i>` represents `<i3>`, `<i8>`, or `<i16>`.

Example:

```
ldc     1
ldc     -9
ldc     130
             ; [1, -9, 130, ...
```

ldv Load from Local Variable

Assembler Syntax: `ldv <u3> or <u8>`

Description: Retrieves a value or a reference from a local variable and pushes it onto the operand stack. A function parameter is also considered as a local variable (see ordering in the `enter`/`ret` instructions). This instruction has one operand, `u3` or `u8`, which indicates the variable number in the current stack frame to push.

Operation:

```
void ldv(u8 localVarNumber) {
    push(stack[localVarNumber]);
}
```

Stack Before: `[...`

Stack After: `[v, ...`

Example:

```
public void fct() {
    int n;          // local variable 0

    // ...

          ldv     0
                        ; [v, ...
          trap    0
                        ; [...
}
```

mul

Multiply

Assembler Syntax: `mul`

Description: The `mul` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 * v2` and is pushed back onto the operand stack.

Operation:

```
void mul() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 * v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
                ; [...
ldc     2
                ; [2, ...
ldc     -3
                ; [2, -3, ...
mul
                ; [-6, ...
```

neg

Negate

Assembler Syntax: `neg`

Description: The `neg` pops the value `v` from the operand stack. The result `r` is `-v`, the bitwise two's complement of `v`, and is pushed back onto the operand stack.

Operation:

```
void neg() {
    stack[sp] = (i32)-stack[sp];
}
```

Stack Before: `[v, ...`

Stack After: `[-v, ...`

Example:

```
                ; [...
ldc     9
                ; [9, ...
neg
                ; [-9, ...
```

not Bitwise One's Complement

Assembler Syntax: `not`

Description: The `not` pops the value `v` from the operand stack. The result `r` is `~v`, the bitwise one's complement of `v`,and is pushed back onto the operand stack.

Operation:

```
void not() {
    stack[sp] = (i32)~stack[sp];
}
```

Stack Before: `[...`

Stack After: `[...`

Example:

```
                ; [...
ldc     0xAA55
                ; [0xAA55, ...      or [0b1010101001010101, ...
not
                ; [0x55AA, ...      or [0b0101010110101010, ...
```

or

Bitwise Or

Assembler Syntax: `or`

Description: The `or` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 | v2` and is pushed back onto the operand stack.

Operation:

```
void or() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 | v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
              ; [...
ldc     0b0101
              ; [5, ...
ldc     0b0110
              ; [5, 4, ...
or
              ; [7, ...
```

pop

Remove Top of Stack

Assembler Syntax: `pop`

Description: Discards the top of stack.

Operation:

```
void pop() { --sp; }
```

Stack Before: `[v, ...`

Stack After: `[...`

Example:

```
                ; [...
    ldc     5
                ; [5, ...
    pop
                ; [...
```

rem Remainder

Assembler Syntax: `rem`

Description: The `rem` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 % v2` and is pushed back onto the operand stack.

Operation:

```
void rem() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 % v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
              ; [...
ldc     3
              ; [3, ...
ldc     2
              ; [3, 2, ...
rem
              ; [1, ...
```

ret Clean up Frame and Return

Assembler Syntax: `ret`

Description: The `ret` instruction is the last instruction of a function. It returns and restores the frame context of its caller, and sets up the context of the current function. The operand `u4` is divided into two fields of values containing a flag `v` if the function returns a value or not (`void`), and the number of local variables that has been allocated within the function (`nl`).

Operation:

```
 3 2 1 0
+-+-----+
|v|  nl |  if v = 0 means the operand stack contains no return value (void)
+-+-----+     v = 1 means the operand stack contains a value to be returned
    u4
```

```
void ret(int u4) {
    int   v  = (u4 >> 3) & 0x01;
    int   nl =  u4      & 0x07;
    int   (*retAddr)();

    if (v) v = stack[sp--]; // save the return value in v (if any)

    sp -= nl;               // deallocate space for local variables
    bp = stack[sp--];       // pop (restore) caller's bp (context)
    retAddr = stack[sp--];  // pop (save) caller's return address
    bp = sp;                // set frame context for the current function
    sp += nl;               // allocate space for local variables
}

void ret() {
    int   u5 = stack[bp-2];
    int   v  = (u5 >> 4) & 0x01;
    int   np = (u5 >> 2) & 0x03;
    int   nl =  u5      & 0x03;

    int   (*retAddr)();
    int   retVal;

    if (v) retVal = stack[sp--]; // save the return value in v (if any)
    bp = stack[sp--];            // pop (restore) caller's bp (context)
    retAddr = stack[sp--];       // pop (save) caller's return address
    sp -= (np+nl+1);             // deallocate space for parameters, local variables, and <u5>
    if (v) stack[++sp] = retVal; // push back the return value (if any)
    stack[++sp] = retAddr;       // push back the caller's return address
}
```

Stack Before:

```
    sp->|  retVal | (if any)
    bp->|caller bp| bp + 0
        | retAddr | bp - 1
        |   <u5>  | bp - 2
        |   ln-1  | bp - 3
        ~   ...   ~   ...
        |   l1    |
        |   l0    |
        |   pn-1  |
        ~   ...   ~   ...
        |   p1    |
        |   p0    |
        +---------+

        [p0, p1, ..., pn-1, l0, l1, ..., ln-1, <u5>, ra, bp, ...

   sp -> |  retVal | (if any)
         |   ln-1  |
         ~   ...   ~   ...
         |   l1    | bp + 2
         |   l0    | bp + 1
   bp -> |caller bp| bp + 0
         | retAddr | bp - 1
         |   pn-1  | bp - 2
         ~   ...   ~   ...
         |   p1    |
         |   p0    |
         +---------+

         [p0, p1, ..., pn-1, ra, bp, l0, l1, ..., ln-1, ...
```

Stack After:

```
    sp->| retAddr |
    sp->|  retVal | (if any)
        +---------+

        [p0, p1, ..., pn-1, l0, l1, ..., ln-1, <u5>, ra, bp, ...


   sp -> |   ln-1  |
         ~   ...   ~   ...
         |   l1    | bp + 2
         |   l0    | bp + 1
   bp -> |caller bp| bp + 0
         | retAddr | bp - 1
         |   pn-1  | bp - 2
         ~   ...   ~   ...
         |   p1    |
         |   p0    |
         +---------+

         [p0, p1, ..., pn-1, ra, bp, l0, l1, ..., ln-1, ...
```

Example: The following is the stack state after the execution of the enter (*):

```
sp -> |   j     | bp + 2
      |   i     | bp + 1
bp -> |caller bp| bp + 0
      | retAddr | bp - 1
      |   p     | bp - 2
      +---------+

      [p, retAddr, bp, i, j, ...


void fct(int p) {     // function with one parameter and two local variables
    int i, j;         // where np = 0x01 and nl = 0x02

                      // enter  6  ; (0x01 << 2)|0x02

                      // (*)

                      //         ;                                bp - getFrameOffset(v,np)
    j = i = p;        // ldv    0 ; load from stack[bp-2] or stack[bp - getFrameOffset(0, 1)]
                      // dup
                      // stv    1 ; store to  stack[bp+1] or stack[bp - getFrameOffset(1, 1)]
                      // stv    2 ; store to  stack[bp+2] or stack[bp - getFramrOffset(2, 1)]

    //...
                      // ret
}
```

shl

Shift Left

Assembler Syntax: `shl`

Description: The `shl` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 << v2` and is pushed back onto the operand stack.

Operation:

```
void shl() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 << v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
                ; [...
    ldc     0b0110
                ; [6, ...
    ldc     1
                ; [6, 1, ...
    shl
                ; [12, ...
```

shr

Shift Right

Assembler Syntax: `shr`

Description: The `shr` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 >> v2` and is pushed back onto the operand stack.

Operation:

```
void shr() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 >> v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
                ; [...
    ldc     0b0110
                ; [6, ...
    ldc     1
                ; [6, 1, ...
    shr
                ; [3, ...
```

sub Substract

Assembler Syntax: `sub`

Description: The `sub` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 - v2` and is pushed back onto the operand stack.

Operation:

```
void sub() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 - v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

stv Store into Local Variable

Assembler Syntax: `stv <u3> or <u8>`

Description: Pops a value or a reference from the operand stack and stores it in a parameter or a local variable. A function parameter is also considered as a local variable (see ordering in the `enter`/`ret` instructions). This instruction has one operand, `u3` or `u8`, which indicates the variable number in the current stack frame to push.

Operation:

```
void stv(u8 localVarNumber) {
    i32 value = pop();

    stack[localVarNumber] = value;
}
```

Stack Before: `[v, ...`

Stack After: `[...`

Example:

```
public void fct() {
    int n;          // local variable 0

    // ...

    n = 3;

          ldc     3
                       ; [3, ...
          stv     0
                       ; [...
}
```

teq — Test for Equality

Assembler Syntax: `teq`

Description: The `teq` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 == v2` and is pushed back onto the operand stack.

Operation:

```
void teq() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 == v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

tge Test for Greater or Equal

Assembler Syntax: `tge`

Description: The `tge` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 >= v2` and is pushed back onto the operand stack.

Operation:

```
void tge() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 >= v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

tgt Test for Greater Than

Assembler Syntax: `tgt`

Description: The `tgt` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 > v2` and is pushed back onto the operand stack.

Operation:

```
void tgt() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 > v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

tle Test for Less Than or Equal

Assembler Syntax: `tle`

Description: The `tle` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 <= v2` and is pushed back onto the operand stack.

Operation:

```
void tle() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 <= v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

tlt Test for Less Than

Assembler Syntax: `add`

Description: The `tlt` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 < v2` and is pushed back onto the operand stack.

Operation:

```
void tlt() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 < v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

tne Test for Non Equality

Assembler Syntax: `tne`

Description: The `tne` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 != v2` and is pushed back onto the operand stack.

Operation:

```
void tne() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 != v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

trap

Trap

Assembler Syntax: `trap <u8>`

Description: The `trap` instruction provides customized services for developers. In other words, its behavior can be defined for the need of the embedded target application. This instruction has one operand `u8` which indicates the service number requested.

The current **Cm VM** makes 8 services available for console output services (debugging purpose). In our case, the behavior of the `trap` pops the value `v` from the operand stack and prints the value on the console output.

The complete implementation of the `trap` instructions below are isolated in the `system.h` and `system.c` files with the source code of the **Cm VM**. Developers can replace these services with their own implementations.

Operation:

```
trap 0x82 (PutI)  - Print a signed integer (int) on console output.
trap 0x83 (PutU)  - Print an unsigned integer (uint) on console output.
trap 0x81 (PutC)  - Print a character (char) on console output.
trap 0x80 (PutB)  - Print a boolean (bool) on console output.
trap 0x86 (PutX)  - Print a byte (u8) on console output. The byte
                    is converted to two hexadecimal digits.
trap 0x85 (PutS)  - Print a C string on console output.
trap 0x87 (PutN)  - Print a newline on console output.
```

Stack Before: `[v, ...`

Stack After: `[...`

Example:

xor Bitwise Exclusive Or

Assembler Syntax: `xor`

Description: The `xor` pops the values `v1` and `v2` from the operand stack. The result `r` is `v1 ^ v2` and is pushed back onto the operand stack.

Operation:

```
void xor() {
    i32 v2 = pop();
    i32 v1 = stack[sp];

    stack[sp] = (i32)(v1 ^ v2);
}
```

Stack Before: `[v1, v2, ...`

Stack After: `[r, ...`

Example:

```
              ; [...
ldc    0b0101
              ; [5, ...
ldc    0b0110
              ; [5, 4, ...
xor
              ; [3, ...
```

Chapter 2

Code Patterns: A Suite of Pre-compiled Test Programs

In this chapter, we present a suite of test programs and their corresponding code generation for the **Cm VM**. The format of the generated programs are in assembly language. These code patterns help to understand the resulting instructions and how they will be interpreted by the **Cm VM**.

2.1 Test 01: Value Types (Literals)

Test Program:

```
void Main() {
    puts("Test 01: Value Types (Literals)\n");
    puts("-128|127|127|127|000DECAF|0000AB8D|0|9|a|A|10|10|10|10|10|false|true\n");

    // Integral literals:
    puti(-128); putc('|'); puti(+127); putc('|');
    puti(127);  putc('|'); putu(127U); putc('|');

    putx(0xDECAF); putc('|'); putx(0XAB8D); putc('|');

    // Character literals:
    putc('0');   putc('|'); putc('9');   putc('|');
    putc('a');   putc('|'); putc('A');   putc('|');
    puti('\n');  putc('|'); puti('\xA'); putc('|');
    puti('\uA'); putc('|'); puti(0xA);   putc('|'); puti(0x00000A); putc('|');

    // Boolean literals:
    putb(false);   putc('|'); putb(true);
    putn();
}
```

Corresponding Assembly Source Code Generation:

```
sAddr Obj. Code    Size Label                                Name       Operand                          Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0099       3                                        br.i16     $Component_End
:0003   [0003]      0   T.C.Main@()v
*0003               0   ;
:0003 D5 00A1       3                                        lda.i16    Test 01: Value Types (Literals)
 0006 FF 85         2                                        trap       85                               ; puts
*0008               0   ;
:0008 D5 00BD       3                                        lda.i16   -128|127|127|127|000DECAF|0000AB8D|0|9|a|A|10|1
 000b FF 85         2                                        trap       85                               ; puts
*000d               0   ;
*000d D9 80         2                                        ldc.i8     -128
 000f FF 82         2                                        trap       82                               ; puti
*0011               0   ;
*0011 D9 7C         2                                        ldc.i8     124
 0013 FF 81         2                                        trap       81                               ; putc
*0015               0   ;
*0015 D9 7F         2                                        ldc.i8     127
 0017 FF 82         2                                        trap       82                               ; puti
*0019               0   ;
*0019 D9 7C         2                                        ldc.i8     124
 001b FF 81         2                                        trap       81                               ; putc
*001d               0   ;
*001d D9 7F         2                                        ldc.i8     127
 001f FF 82         2                                        trap       82                               ; puti
*0021               0   ;
*0021 D9 7C         2                                        ldc.i8     124
 0023 FF 81         2                                        trap       81                               ; putc
*0025               0   ;
*0025 D9 7F         2                                        ldc.i8     127
 0027 FF 83         2                                        trap       83                               ; putu
*0029               0   ;
*0029 D9 7C         2                                        ldc.i8     124
 002b FF 81         2                                        trap       81                               ; putc
*002d               0   ;
*002d DB 000DECAF   5                                        ldc.i32    912559
 0032 FF 86         2                                        trap       86                               ; putx
*0034               0   ;
*0034 D9 7C         2                                        ldc.i8     124
 0036 FF 81         2                                        trap       81                               ; putc
*0038               0   ;
*0038 DB 0000AB8D   5                                        ldc.i32    43917
 003d FF 86         2                                        trap       86                               ; putx
*003f               0   ;
*003f D9 7C         2                                        ldc.i8     124
 0041 FF 81         2                                        trap       81                               ; putc
*0043               0   ;
*0043 D9 30         2                                        ldc.i8     48
 0045 FF 81         2                                        trap       81                               ; putc
*0047               0   ;
*0047 D9 7C         2                                        ldc.i8     124
 0049 FF 81         2                                        trap       81                               ; putc
*004b               0   ;
*004b D9 39         2                                        ldc.i8     57
 004d FF 81         2                                        trap       81                               ; putc
*004f               0   ;
*004f D9 7C         2                                        ldc.i8     124
 0051 FF 81         2                                        trap       81                               ; putc
*0053               0   ;
*0053 D9 61         2                                        ldc.i8     97
 0055 FF 81         2                                        trap       81                               ; putc
*0057               0   ;
```

```
*0057 D9 7C         2                                       ldc.i8      124
 0059 FF 81         2                                       trap        81                          ; putc
*005b               0   ;
*005b D9 41         2                                       ldc.i8      65
 005d FF 81         2                                       trap        81                          ; putc
*005f               0   ;
*005f D9 7C         2                                       ldc.i8      124
 0061 FF 81         2                                       trap        81                          ; putc
*0063               0   ;
*0063 D9 0A         2                                       ldc.i8      10
 0065 FF 82         2                                       trap        82                          ; puti
*0067               0   ;
*0067 D9 7C         2                                       ldc.i8      124
 0069 FF 81         2                                       trap        81                          ; putc
*006b               0   ;
*006b D9 0A         2                                       ldc.i8      10
 006d FF 82         2                                       trap        82                          ; puti
*006f               0   ;
*006f D9 7C         2                                       ldc.i8      124
 0071 FF 81         2                                       trap        81                          ; putc
*0073               0   ;
*0073 D9 0A         2                                       ldc.i8      10
 0075 FF 82         2                                       trap        82                          ; puti
*0077               0   ;
*0077 D9 7C         2                                       ldc.i8      124
 0079 FF 81         2                                       trap        81                          ; putc
*007b               0   ;
*007b D9 0A         2                                       ldc.i8      10
 007d FF 82         2                                       trap        82                          ; puti
*007f               0   ;
*007f D9 7C         2                                       ldc.i8      124
 0081 FF 81         2                                       trap        81                          ; putc
*0083               0   ;
*0083 D9 0A         2                                       ldc.i8      10
 0085 FF 82         2                                       trap        82                          ; puti
*0087               0   ;
*0087 D9 7C         2                                       ldc.i8      124
 0089 FF 81         2                                       trap        81                          ; putc
*008b               0   ;
*008b 90            1                                       ldc.i3      0
 008c FF 80         2                                       trap        80                          ; putb
*008e               0   ;
*008e D9 7C         2                                       ldc.i8      124
 0090 FF 81         2                                       trap        81                          ; putc
*0092               0   ;
*0092 91            1                                       ldc.i3      1
 0093 FF 80         2                                       trap        80                          ; putb
*0095               0   ;
 0095 FF 87         2                                       trap        87                          ; putn
*0097               0   ;
*0097 04            1                                       ret
:0098   [0152]      0   T.C._init@()v
*0098 04            1                                       ret
:0099   [0153]      0   $Component_End
:0099 E7 FFFF       3                                       calls.i16   T.C._init@()v
:009c E7 FF67       3                                       calls.i16   T.C.Main@()v
*009f 00            1                                       halt
/00a0 54 54 ..      4   T.C                                 .cstring    "T.C"
/00a4 54 54 ..     33   Test 01: Value Types (Literals)     .cstring    "Test 01: Value Types (Literals)"
/00c5 2D 2D ..     70   -128|127|127|127|000DECAF|0000AB8D|0|9|a|A|10|10|10|10|10|false|true
                                                            .cstring    "-128|127|127|127|000DECAF|0000AB8D|0|9|a|A|10
Generate 'exe' file 'T01.exe' with 267 bytes
```

2.2 Test 02: Conditional Operator

Test Program:

```
void Main() {
    puts("Test 02: Conditional Operator\n");
    puts("3|4|5\n");

    var int a, b, r;

    a = 3; b = 4;
    r = a < b ? a : b;
    puti(r); putc('|'); // 3

    a = -4;
    r = a < 0 ? -a : a;
    puti(r); putc('|'); // 4

    a = 5;
    r = a < 0 ? -a : a;
    puti(r);            // 5

    putn();
}
```

Corresponding Assembly Source Code Generation:

```
sAddr Obj. Code    Size Label                         Name      Operand                      Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 004D       3                                 br.i16    $Component_End
:0003   [0003]      0   T.C.Main@()v
*0003 73            1                                 enter     3
*0004               0   ;
:0004 D5 0054       3                                 lda.i16   Test 02: Conditional Operator
 0007 FF 85         2                                 trap      85                           ; puts
*0009               0   ;
:0009 D5 006E       3                                 lda.i16   3|4|5
 000c FF 85         2                                 trap      85                           ; puts
*000e               0   ;
*000e               0   ;
*000e 93            1                                 ldc.i3    3
*000f A8            1                                 stv.u3    0
*0010               0   ;
*0010 D9 04         2                                 ldc.i8    4
*0012 A9            1                                 stv.u3    1
*0013               0   ;
*0013 A0            1                                 ldv.u3    0
*0014 A1            1                                 ldv.u3    1
*0015 1C            1                                 tlt
:0016 E3 05         2                                 brf.i8    $1
*0018 A0            1                                 ldv.u3    0
:0019 E0 03         2                                 br.i8     $2
:001b   [0027]      0   $1
*001b A1            1                                 ldv.u3    1
:001c   [0028]      0   $2
*001c AA            1                                 stv.u3    2
*001d               0   ;
*001d A2            1                                 ldv.u3    2
 001e FF 82         2                                 trap      82                           ; puti
*0020               0   ;
*0020 D9 7C         2                                 ldc.i8    124
 0022 FF 81         2                                 trap      81                           ; putc
*0024               0   ;
*0024 94            1                                 ldc.i3    -4
*0025 A8            1                                 stv.u3    0
*0026               0   ;
*0026 A0            1                                 ldv.u3    0
*0027 90            1                                 ldc.i3    0
*0028 1C            1                                 tlt
:0029 E3 06         2                                 brf.i8    $3
*002b A0            1                                 ldv.u3    0
*002c 10            1                                 neg
:002d E0 03         2                                 br.i8     $4
:002f   [0047]      0   $3
*002f A0            1                                 ldv.u3    0
:0030   [0048]      0   $4
*0030 AA            1                                 stv.u3    2
*0031               0   ;
*0031 A2            1                                 ldv.u3    2
 0032 FF 82         2                                 trap      82                           ; puti
*0034               0   ;
*0034 D9 7C         2                                 ldc.i8    124
 0036 FF 81         2                                 trap      81                           ; putc
*0038               0   ;
*0038 D9 05         2                                 ldc.i8    5
*003a A8            1                                 stv.u3    0
*003b               0   ;
*003b A0            1                                 ldv.u3    0
*003c 90            1                                 ldc.i3    0
```

```
*003d 1C            1                                  tlt
:003e E3 06         2                                  brf.i8     $5
*0040 A0            1                                  ldv.u3     0
*0041 10            1                                  neg
:0042 E0 03         2                                  br.i8      $6
:0044   [0068]      0   $5
*0044 A0            1                                  ldv.u3     0
:0045   [0069]      0   $6
*0045 AA            1                                  stv.u3     2
*0046               0   ;
*0046 A2            1                                  ldv.u3     2
 0047 FF 82         2                                  trap       82                              ; puti
*0049               0   ;
 0049 FF 87         2                                  trap       87                              ; putn
*004b               0   ;
*004b 03            1                                  exit
:004c   [0076]      0   T.C._init@()v
*004c 04            1                                  ret
:004d   [0077]      0   $Component_End
:004d E7 FFFF       3                                  calls.i16  T.C._init@()v
:0050 E7 FFB3       3                                  calls.i16  T.C.Main@()v
*0053 00            1                                  halt
/0054 54 54 ..      4   T.C                            .cstring   "T.C"
/0058 54 54 ..     31   Test 02: Conditional Operator  .cstring   "Test 02: Conditional Operator"
                                                       .cstring   "3|4|5"

Generate 'exe' file 'T02.exe' with 126 bytes
```

2.3 Test 03: Bitwise Operators

Test Program:

```
void Main() {
    puts("Test 03: Bitwise Operators\n");
    puts("0000005A|00003C5A|00003C00|FFFFFFA5|FFFFC3A5\n");

    var int a, b, r;

    a = 0x0000005A;     // = 00000000 00000000 00000000 01011010
    b = 0x00003C5A;     // = 00000000 00000000 00111100 01011010

                // Result:
    r = a & b; // 0000005A = 00000000 00000000 00000000 01011010
    putx(r); putc('|');

    r = a | b; // 00003C5A = 00000000 00000000 00111100 01011010
    putx(r); putc('|');

    r = a ^ b; // 00003C00 = 00000000 00000000 00111100 00000000
    putx(r); putc('|');

    r = ~a;    // FFFFFFA5 = 11111111 11111111 11111111 10100101
    putx(r); putc('|');

    r = ~b;    // FFFFC3A5 = 11111111 11111111 11000011 10100101
    putx(r);
    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                             Name        Operand                            Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 004C       3                                     br.i16      $Component_End
:0003   [0003]      0   T.Expr.Main@()v
*0003 73            1                                     enter       3
*0004               0   ;
:0004 D5 0056       3                                     lda.i16     Test 03: Bitwise Operators
 0007 FF 85         2                                     trap        85                                 ; puts
*0009               0   ;
:0009 D5 006D       3                                     lda.i16     0000005A|00003C5A|00003C00|FFFFFFA5|FFFFC3A5
 000c FF 85         2                                     trap        85                                 ; puts
*000e               0   ;
*000e D9 5A         2                                     ldc.i8      90
*0010 A8            1                                     stv.u3      0
*0011               0   ;
*0011 DA 3C5A       3                                     ldc.i16     15450
*0014 A9            1                                     stv.u3      1
*0015               0   ;
*0015 A0            1                                     ldv.u3      0
*0016 A1            1                                     ldv.u3      1
*0017 0D            1                                     and
*0018 AA            1                                     stv.u3      2
*0019               0   ;
*0019 A2            1                                     ldv.u3      2
 001a FF 86         2                                     trap        86                                 ; putx
*001c               0   ;
*001c D9 7C         2                                     ldc.i8      124
 001e FF 81         2                                     trap        81                                 ; putc
*0020               0   ;
*0020 A0            1                                     ldv.u3      0
*0021 A1            1                                     ldv.u3      1
*0022 0E            1                                     or
*0023 AA            1                                     stv.u3      2
*0024               0   ;
*0024 A2            1                                     ldv.u3      2
 0025 FF 86         2                                     trap        86                                 ; putx
*0027               0   ;
*0027 D9 7C         2                                     ldc.i8      124
 0029 FF 81         2                                     trap        81                                 ; putc
*002b               0   ;
*002b A0            1                                     ldv.u3      0
*002c A1            1                                     ldv.u3      1
*002d 0F            1                                     xor
*002e AA            1                                     stv.u3      2
*002f               0   ;
*002f A2            1                                     ldv.u3      2
 0030 FF 86         2                                     trap        86                                 ; putx
*0032               0   ;
*0032 D9 7C         2                                     ldc.i8      124
 0034 FF 81         2                                     trap        81                                 ; putc
*0036               0   ;
*0036 A0            1                                     ldv.u3      0
*0037 97            1                                     ldc.i3      -1
*0038 0F            1                                     xor
*0039 AA            1                                     stv.u3      2
*003a               0   ;
*003a A2            1                                     ldv.u3      2
 003b FF 86         2                                     trap        86                                 ; putx
*003d               0   ;
*003d D9 7C         2                                     ldc.i8      124
 003f FF 81         2                                     trap        81                                 ; putc
*0041               0   ;
```

```
*0041 A1           1                                    ldv.u3     1
*0042 97           1                                    ldc.i3     -1
*0043 0F           1                                    xor
*0044 AA           1                                    stv.u3     2
*0045              0   ;
*0045 A2           1                                    ldv.u3     2
 0046 FF 86        2                                    trap       86                          ; putx
*0048              0   ;
 0048 FF 87        2                                    trap       87                          ; putn
*004a              0   ;
*004a 03           1                                    exit
:004b   [0075]     0   T.Expr._init@()v
*004b 04           1                                    ret
:004c   [0076]     0   $Component_End
:004c E7 FFFF      3                                    calls.i16  T.Expr._init@()v
:004f E7 FFB4      3                                    calls.i16  T.Expr.Main@()v
*0052 00           1                                    halt
/0053 54 54 ..     7   T.Expr                           .cstring   "T.Expr"
/005a 54 54 ..    28   Test 03: Bitwise Operators       .cstring   "Test 03: Bitwise Operators"
                                                        .cstring   "0000005A|00003C5A|00003C00|FFFFFFA5|FFFFC3A5"
Generate 'exe' file 'T03.exe' with 164 bytes
```

2.4 Test 04: Equality Operators

Test Program:

```
void Main() {
    puts("Test 04: Equality Operators\n");
    puts("false|true\n");

    var int i;
    var bool r;    // Result.

    i = '9';

    r = i == 9;    // false
    putb(r); putc('|');

    r = i != 9;    // true
    putb(r);
    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                          Name        Operand                         Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0035       3                                  br.i16      $Component_End
:0003   [0003]      0   T.Expr.Main@()v
*0003 72            1                                  enter       2
*0004               0   ;
:0004 D5 003F       3                                  lda.i16     Test 04: Equality Operators
 0007 FF 85         2                                  trap        85                              ; puts
*0009               0   ;
:0009 D5 0057       3                                  lda.i16     false|true
 000c FF 85         2                                  trap        85                              ; puts
*000e               0   ;
*000e D9 39         2                                  ldc.i8      57
*0010 A8            1                                  stv.u3      0
*0011               0   ;
*0011 A0            1                                  ldv.u3      0
*0012 D9 09         2                                  ldc.i8      9
*0014 1B            1                                  tne
:0015 E3 05         2                                  brf.i8      $1
*0017 90            1                                  ldc.i3      0
:0018 E0 03         2                                  br.i8       $2
:001a   [0026]      0   $1
*001a 91            1                                  ldc.i3      1
:001b   [0027]      0   $2
*001b A9            1                                  stv.u3      1
*001c               0   ;
*001c A1            1                                  ldv.u3      1
 001d FF 80         2                                  trap        80                              ; putb
*001f               0   ;
*001f D9 7C         2                                  ldc.i8      124
 0021 FF 81         2                                  trap        81                              ; putc
*0023               0   ;
*0023 A0            1                                  ldv.u3      0
*0024 D9 09         2                                  ldc.i8      9
*0026 1A            1                                  teq
:0027 E3 05         2                                  brf.i8      $3
*0029 90            1                                  ldc.i3      0
:002a E0 03         2                                  br.i8       $4
:002c   [0044]      0   $3
*002c 91            1                                  ldc.i3      1
:002d   [0045]      0   $4
*002d A9            1                                  stv.u3      1
*002e               0   ;
*002e A1            1                                  ldv.u3      1
 002f FF 80         2                                  trap        80                              ; putb
*0031               0   ;
 0031 FF 87         2                                  trap        87                              ; putn
*0033               0   ;
*0033 03            1                                  exit
:0034   [0052]      0   T.Expr._init@()v
*0034 04            1                                  ret
:0035   [0053]      0   $Component_End
:0035 E7 FFFF       3                                  calls.i16   T.Expr._init@()v
:0038 E7 FFCB       3                                  calls.i16   T.Expr.Main@()v
*003b 00            1                                  halt
/003c 54 54 ..      7   T.Expr                         .cstring    "T.Expr"
/0043 54 54 ..     29   Test 04: Equality Operators    .cstring    "Test 04: Equality Operators"
/0060 66 66 ..     12   false|true                     .cstring    "false|true"

Generate 'exe' file 'T04.exe' with 108 bytes
```

2.5 Test 05: Relational Operators

Test Program:

```
void Main() {
    puts("Test 05: Relational Operators\n");
    puts("true|true|false|false\n");

    var int  a, b;  // Signed integer operands.
    var bool r;     // Result.

    a = 1; b = 2;
    r = a < b;      putb(r); putc('|'); // true|

    a = 3; b = 4;
    r = a <= b;     putb(r); putc('|'); // true|

    a = 5; b = 6;
    r = a > b;      putb(r); putc('|'); // false|

    a = 7; b = 8;
    r = a >= b;     putb(r);            // false|

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                               Name        Operand                       Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0067       3                                       br.i16      $Component_End
:0003   [0003]      0   T.Expr.Main@()v
*0003 73            1                                       enter       3
*0004               0   ;
:0004 D5 0071       3                                       lda.i16     Test 05: Relational Operators
 0007 FF 85         2                                       trap        85                            ; puts
*0009               0   ;
:0009 D5 008B       3                                       lda.i16     true|true|false|false
 000c FF 85         2                                       trap        85                            ; puts
*000e               0   ;
*000e 91            1                                       ldc.i3      1
*000f A8            1                                       stv.u3      0
*0010               0   ;
*0010 92            1                                       ldc.i3      2
*0011 A9            1                                       stv.u3      1
*0012               0   ;
*0012 A0            1                                       ldv.u3      0
*0013 A1            1                                       ldv.u3      1
*0014 1F            1                                       tge
:0015 E3 05         2                                       brf.i8      $1
*0017 90            1                                       ldc.i3      0
:0018 E0 03         2                                       br.i8       $2
:001a   [0026]      0   $1
*001a 91            1                                       ldc.i3      1
:001b   [0027]      0   $2
*001b AA            1                                       stv.u3      2
*001c               0   ;
*001c A2            1                                       ldv.u3      2
 001d FF 80         2                                       trap        80                            ; putb
*001f               0   ;
*001f D9 7C         2                                       ldc.i8      124
 0021 FF 81         2                                       trap        81                            ; putc
*0023               0   ;
*0023 93            1                                       ldc.i3      3
*0024 A8            1                                       stv.u3      0
*0025               0   ;
*0025 D9 04         2                                       ldc.i8      4
*0027 A9            1                                       stv.u3      1
*0028               0   ;
*0028 A0            1                                       ldv.u3      0
*0029 A1            1                                       ldv.u3      1
*002a 1D            1                                       tgt
:002b E3 05         2                                       brf.i8      $3
*002d 90            1                                       ldc.i3      0
:002e E0 03         2                                       br.i8       $4
:0030   [0048]      0   $3
*0030 91            1                                       ldc.i3      1
:0031   [0049]      0   $4
*0031 AA            1                                       stv.u3      2
*0032               0   ;
*0032 A2            1                                       ldv.u3      2
 0033 FF 80         2                                       trap        80                            ; putb
*0035               0   ;
*0035 D9 7C         2                                       ldc.i8      124
 0037 FF 81         2                                       trap        81                            ; putc
*0039               0   ;
*0039 D9 05         2                                       ldc.i8      5
*003b A8            1                                       stv.u3      0
*003c               0   ;
*003c D9 06         2                                       ldc.i8      6
```

```
*003e A9           1                                   stv.u3     1
*003f              0   ;
*003f A0           1                                   ldv.u3     0
*0040 A1           1                                   ldv.u3     1
*0041 1E           1                                   tle
:0042 E3 05        2                                   brf.i8     $5
*0044 90           1                                   ldc.i3     0
:0045 E0 03        2                                   br.i8      $6
:0047   [0071]     0   $5
*0047 91           1                                   ldc.i3     1
:0048   [0072]     0   $6
*0048 AA           1                                   stv.u3     2
*0049              0   ;
*0049 A2           1                                   ldv.u3     2
 004a FF 80        2                                   trap       80                                ; putb
*004c              0   ;
*004c D9 7C        2                                   ldc.i8     124
 004e FF 81        2                                   trap       81                                ; putc
*0050              0   ;
*0050 D9 07        2                                   ldc.i8     7
*0052 A8           1                                   stv.u3     0
*0053              0   ;
*0053 D9 08        2                                   ldc.i8     8
*0055 A9           1                                   stv.u3     1
*0056              0   ;
*0056 A0           1                                   ldv.u3     0
*0057 A1           1                                   ldv.u3     1
*0058 1C           1                                   tlt
:0059 E3 05        2                                   brf.i8     $7
*005b 90           1                                   ldc.i3     0
:005c E0 03        2                                   br.i8      $8
:005e   [0094]     0   $7
*005e 91           1                                   ldc.i3     1
:005f   [0095]     0   $8
*005f AA           1                                   stv.u3     2
*0060              0   ;
*0060 A2           1                                   ldv.u3     2
 0061 FF 80        2                                   trap       80                                ; putb
*0063              0   ;
 0063 FF 87        2                                   trap       87                                ; putn
*0065              0   ;
*0065 03           1                                   exit
:0066   [0102]     0   T.Expr._init@()v
*0066 04           1                                   ret
:0067   [0103]     0   $Component_End
:0067 E7 FFFF      3                                   calls.i16  T.Expr._init@()v
:006a E7 FF99      3                                   calls.i16  T.Expr.Main@()v
*006d 00           1                                   halt
/006e 54 54 ..     7   T.Expr                          .cstring   "T.Expr"
/0075 54 54 ..    31   Test 05: Relational Operators   .cstring   "Test 05: Relational Operators"
/0094 74 74 ..    23   true|true|false|false           .cstring   "true|true|false|false"

Generate 'exe' file 'T05.exe' with 171 bytes
```

2.6 Test 06: Shift Operators

Test Program:

```
void Main() {
    puts("Test 06: Shift Operators\n");
    puts("FFFFFFA6|FFFFFFD3|0000F168|00001E2D"); putn();

    var int a, b, r;

    a = 0x0000005A; // = 00000000 00000000 00000000 01011010
    b = 0x00003C5A; // = 00000000 00000000 00111100 01011010

                    // Result.
    a = -a;         // FFFFFFA6 = 11111111 11111111 11111111 10100101
    putx(a); putc('|');
    r = a >> 1;     // FFFFFFD3 = 11111111 11111111 11111111 11010010
    putx(r); putc('|');
    r = b << 2;     // 0000F168 = 00000000 00000000 11110001 01101000
    putx(r); putc('|');
    r = r >> 3;     // 00001E2D = 00000000 00000000 00011110 00101101
    putx(r);

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                              Name        Operand                              Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0042       3                                      br.i16      $Component_End
:0003   [0003]      0   T.Expr.Main@()v
*0003 73            1                                      enter       3
*0004               0   ;
:0004 D5 004C       3                                      lda.i16     Test 06: Shift Operators
 0007 FF 85         2                                      trap        85                                   ; puts
*0009               0   ;
:0009 D5 0061       3                                      lda.i16     FFFFFFA6|FFFFFFD3|0000F168|00001E2D
 000c FF 85         2                                      trap        85                                   ; puts
*000e               0   ;
 000e FF 87         2                                      trap        87                                   ; putn
*0010               0   ;
*0010 D9 5A         2                                      ldc.i8      90
*0012 A8            1                                      stv.u3      0
*0013               0   ;
*0013 DA 3C5A       3                                      ldc.i16     15450
*0016 A9            1                                      stv.u3      1
*0017               0   ;
*0017 A0            1                                      ldv.u3      0
*0018 10            1                                      neg
*0019 A8            1                                      stv.u3      0
*001a               0   ;
*001a A0            1                                      ldv.u3      0
 001b FF 86         2                                      trap        86                                   ; putx
*001d               0   ;
*001d D9 7C         2                                      ldc.i8      124
 001f FF 81         2                                      trap        81                                   ; putc
*0021               0   ;
*0021 A0            1                                      ldv.u3      0
*0022 91            1                                      ldc.i3      1
*0023 19            1                                      shr
*0024 AA            1                                      stv.u3      2
*0025               0   ;
*0025 A2            1                                      ldv.u3      2
 0026 FF 86         2                                      trap        86                                   ; putx
*0028               0   ;
*0028 D9 7C         2                                      ldc.i8      124
 002a FF 81         2                                      trap        81                                   ; putc
*002c               0   ;
*002c A1            1                                      ldv.u3      1
*002d 92            1                                      ldc.i3      2
*002e 18            1                                      shl
*002f AA            1                                      stv.u3      2
*0030               0   ;
*0030 A2            1                                      ldv.u3      2
 0031 FF 86         2                                      trap        86                                   ; putx
*0033               0   ;
*0033 D9 7C         2                                      ldc.i8      124
 0035 FF 81         2                                      trap        81                                   ; putc
*0037               0   ;
*0037 A2            1                                      ldv.u3      2
*0038 93            1                                      ldc.i3      3
*0039 19            1                                      shr
*003a AA            1                                      stv.u3      2
*003b               0   ;
*003b A2            1                                      ldv.u3      2
 003c FF 86         2                                      trap        86                                   ; putx
*003e               0   ;
 003e FF 87         2                                      trap        87                                   ; putn
*0040               0   ;
```

```
*0040 03             1                                            exit
:0041   [0065]       0   T.Expr._init@()v
*0041 04             1                                            ret
:0042   [0066]       0   $Component_End
:0042 E7 FFFF        3                                            calls.i16   T.Expr._init@()v
:0045 E7 FFBE        3                                            calls.i16   T.Expr.Main@()v
*0048 00             1                                            halt
/0049 54 54 ..       7   T.Expr                                   .cstring    "T.Expr"
/0050 54 54 ..      26   Test 06: Shift Operators                 .cstring    "Test 06: Shift Operators"
/006a 46 46 ..      36   FFFFFFA6|FFFFFFD3|0000F168|00001E2D .cstring    "FFFFFFA6|FFFFFFD3|0000F168|00001E2D"

Generate 'exe' file 'T06.exe' with 142 bytes
```

2.7 Test 07: Extended Bitwise Assignment Operators

Test Program:

```
void Main() {
    puts("Test 07: Extended Bitwise Assignment Operators"); putn();
    puts("7FFFFFA6|3FFFFFD3|FFFFFD30"); putn();

    var int r;
    r = 0x7FFFFFA6;     // 7FFFFFA6 = 01111111 11111111 11111111 10100110
    putx(r); putc('|');

    r >>= 1;            // 3FFFFFD3 = 00111111 11111111 11111111 11010011
    putx(r); putc('|');

    r <<= 4;            // FFFFFD30 = 11111111 11111111 11111101 00110000
    putx(r);

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                              Name        Operand                         Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0036       3                                      br.i16      $Component_End
:0003   [0003]      0   T.Expr.Main@()v
*0003 71            1                                      enter       1
*0004               0   ;
:0004 D5 0040       3                                      lda.i16     Test 07: Extended Bitwise Assignment Operators
 0007 FF 85         2                                      trap        85                              ; puts
*0009               0   ;
 0009 FF 87         2                                      trap        87                              ; putn
*000b               0   ;
:000b D5 0068       3                                      lda.i16     7FFFFFA6|3FFFFFD3|FFFFFD30
 000e FF 85         2                                      trap        85                              ; puts
*0010               0   ;
 0010 FF 87         2                                      trap        87                              ; putn
*0012               0   ;
*0012 DB 7FFFFFA6   5                                      ldc.i32     2147483558
*0017 A8            1                                      stv.u3      0
*0018               0   ;
*0018 A0            1                                      ldv.u3      0
 0019 FF 86         2                                      trap        86                              ; putx
*001b               0   ;
*001b D9 7C         2                                      ldc.i8      124
 001d FF 81         2                                      trap        81                              ; putc
*001f               0   ;
*001f A0            1                                      ldv.u3      0
*0020 91            1                                      ldc.i3      1
*0021 19            1                                      shr
*0022 A8            1                                      stv.u3      0
*0023               0   ;
*0023 A0            1                                      ldv.u3      0
 0024 FF 86         2                                      trap        86                              ; putx
*0026               0   ;
*0026 D9 7C         2                                      ldc.i8      124
 0028 FF 81         2                                      trap        81                              ; putc
*002a               0   ;
*002a A0            1                                      ldv.u3      0
*002b D9 04         2                                      ldc.i8      4
*002d 18            1                                      shl
*002e A8            1                                      stv.u3      0
*002f               0   ;
*002f A0            1                                      ldv.u3      0
 0030 FF 86         2                                      trap        86                              ; putx
*0032               0   ;
 0032 FF 87         2                                      trap        87                              ; putn
*0034               0   ;
*0034 03            1                                      exit
:0035   [0053]      0   T.Expr._init@()v
*0035 04            1                                      ret
:0036   [0054]      0   $Component_End
:0036 E7 FFFF       3                                      calls.i16   T.Expr._init@()v
:0039 E7 FFCA       3                                      calls.i16   T.Expr.Main@()v
*003c 00            1                                      halt
/003d 54 54 ..      7   T.Expr                             .cstring    "T.Expr"
/0044 54 54 ..     47   Test 07: Extended Bitwise Operators .cstring   "Test 07: Extended Bitwise Assignment Operators
/0073 37 37 ..     27   7FFFFFA6|3FFFFFD3|FFFFFD30         .cstring    "7FFFFFA6|3FFFFFD3|FFFFFD30"

Generate 'exe' file 'T07.exe' with 142 bytes
```

2.8 Test 08: Prefix and Postfix Operators

Test Program:

```
void Main() {
    puts("Test 08: Prefix and Postfix Operators\n");
    puts("7778798887\n");

    var int a, b;

    b = 6;

    a = ++b;  puti(a); puti(b);
    a = b++;  puti(a); puti(b);
    ++b;      puti(a); puti(b);
    a = --b;  puti(a); puti(b);
    a = b--;  puti(a); puti(b);

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                               Name        Operand                          Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0045       3                                       br.i16      $Component_End
:0003   [0003]      0   T.Expr.Main@()v
*0003 72            1                                       enter       2
*0004               0   ;
:0004 D5 004F       3                                       lda.i16     Test 08: Prefix and Postfix Operators
 0007 FF 85         2                                       trap        85                               ; puts
*0009               0   ;
:0009 D5 0071       3                                       lda.i16     7778798887
 000c FF 85         2                                       trap        85                               ; puts
*000e               0   ;
*000e D9 06         2                                       ldc.i8      6
*0010 A9            1                                       stv.u3      1
*0011               0   ;
*0011 B3 01         2                                       incv.u8     1
*0013 A1            1                                       ldv.u3      1
*0014 A8            1                                       stv.u3      0
*0015               0   ;
*0015 A0            1                                       ldv.u3      0
 0016 FF 82         2                                       trap        82                               ; puti
*0018               0   ;
*0018 A1            1                                       ldv.u3      1
 0019 FF 82         2                                       trap        82                               ; puti
*001b               0   ;
*001b A1            1                                       ldv.u3      1
*001c B3 01         2                                       incv.u8     1
*001e A8            1                                       stv.u3      0
*001f               0   ;
*001f A0            1                                       ldv.u3      0
 0020 FF 82         2                                       trap        82                               ; puti
*0022               0   ;
*0022 A1            1                                       ldv.u3      1
 0023 FF 82         2                                       trap        82                               ; puti
*0025               0   ;
*0025 B3 01         2                                       incv.u8     1
*0027               0   ;
*0027 A0            1                                       ldv.u3      0
 0028 FF 82         2                                       trap        82                               ; puti
*002a               0   ;
*002a A1            1                                       ldv.u3      1
 002b FF 82         2                                       trap        82                               ; puti
*002d               0   ;
*002d B4 01         2                                       decv.u8     1
*002f A1            1                                       ldv.u3      1
*0030 A8            1                                       stv.u3      0
*0031               0   ;
*0031 A0            1                                       ldv.u3      0
 0032 FF 82         2                                       trap        82                               ; puti
*0034               0   ;
*0034 A1            1                                       ldv.u3      1
 0035 FF 82         2                                       trap        82                               ; puti
*0037               0   ;
*0037 A1            1                                       ldv.u3      1
*0038 B4 01         2                                       decv.u8     1
*003a A8            1                                       stv.u3      0
*003b               0   ;
*003b A0            1                                       ldv.u3      0
 003c FF 82         2                                       trap        82                               ; puti
*003e               0   ;
*003e A1            1                                       ldv.u3      1
 003f FF 82         2                                       trap        82                               ; puti
```

```
*0041              0   ;
 0041 FF 87        2                                    trap        87                            ; putn
*0043              0   ;
*0043 03           1                                    exit
:0044   [0068]     0   T.Expr._init@()v
*0044 04           1                                    ret
:0045   [0069]     0   $Component_End
:0045 E7 FFFF      3                                    calls.i16   T.Expr._init@()v
:0048 E7 FFBB      3                                    calls.i16   T.Expr.Main@()v
*004b 00           1                                    halt
/004c 54 54 ..     7   T.Expr                           .cstring    "T.Expr"
/0053 54 54 ..    39   Test 08: Prefix Postfix Operators   .cstring    "Test 08: Prefix and Postfix Operators"
/007a 37 37 ..    12   7778798887                       .cstring    "7778798887"

Generate 'exe' file 'T08.exe' with 134 bytes
```

2.9 Test 09: if-else Statement

Test Program:

```
const int  Min = 0;
const int  Max = 9;

int Tick(int count, bool directionUp) {
    if (directionUp) { // If with an else clause.
        if (++count > Max) { // Nested if without an else clause.
            count = Min;
        }
    } else {           // Else clause of the outer if statement.
        if (--count < Min) { // Nested if without an else clause.
            count = Max;
        }
    }
    puti(count); putc('|');
    return count;
}

void Main() {
    puts("Test 09: if-else Statement\n");
    puts("9|0|9|0|1|\n");

    var   int  count;
    var   bool directionUp;

    count = 8;
    directionUp = true;

    count = Tick(count, directionUp); // 9
    count = Tick(count, directionUp); // 0

    directionUp = false;

    count = Tick(count, directionUp); // 9

    directionUp = true;

    count = Tick(count, directionUp); // 0
    count = Tick(count, directionUp); // 1

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                                   Name        Operand                          Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 005F       3                                           br.i16      $Component_End
:0003   [0003]      0   T.Stmt.Tick@(ib)i
*0003 88            1                                           enter       24
*0004               0   ;
*0004 A1            1                                           ldv.u3      1
*0005 90            1                                           ldc.i3      0
*0006 1B            1                                           tne
:0007 E3 0E         2                                           brf.i8      $1
*0009               0   ;
*0009 B3 00         2                                           incv.u8     0
*000b A0            1                                           ldv.u3      0
*000c D9 09         2                                           ldc.i8      9
*000e 1D            1                                           tgt
:000f E3 04         2                                           brf.i8      $2
*0011               0   ;
*0011 90            1                                           ldc.i3      0
*0012 A8            1                                           stv.u3      0
*0013               0   ;
:0013   [0019]      0   $2
*0013               0   ;
:0013 E0 0C         2                                           br.i8       $3
:0015   [0021]      0   $1
*0015               0   ;
*0015 B4 00         2                                           decv.u8     0
*0017 A0            1                                           ldv.u3      0
*0018 90            1                                           ldc.i3      0
*0019 1C            1                                           tlt
:001a E3 05         2                                           brf.i8      $4
*001c               0   ;
*001c D9 09         2                                           ldc.i8      9
*001e A8            1                                           stv.u3      0
*001f               0   ;
:001f   [0031]      0   $4
*001f               0   ;
:001f   [0031]      0   $3
*001f               0   ;
*001f A0            1                                           ldv.u3      0
 0020 FF 82         2                                           trap        82                               ; puti
*0022               0   ;
*0022 D9 7C         2                                           ldc.i8      124
 0024 FF 81         2                                           trap        81                               ; putc
*0026               0   ;
*0026 A0            1                                           ldv.u3      0
*0027 03            1                                           exit
*0028               0   ;
*0028 03            1                                           exit
:0029   [0041]      0   T.Stmt.Main@()v
*0029 72            1                                           enter       2
*002a               0   ;
:002a D5 0043       3                                           lda.i16     Test 09: if-else Statement
 002d FF 85         2                                           trap        85                               ; puts
*002f               0   ;
:002f D5 005A       3                                           lda.i16     9|0|9|0|1|
 0032 FF 85         2                                           trap        85                               ; puts
*0034               0   ;
*0034 D9 08         2                                           ldc.i8      8
*0036 A8            1                                           stv.u3      0
*0037               0   ;
*0037 91            1                                           ldc.i3      1
*0038 A9            1                                           stv.u3      1
```

```
*0039              0   ;
*0039 A0           1                                        ldv.u3     0
*003a A1           1                                        ldv.u3     1
:003b E7 FFC8      3                                        calls.i16  T.Stmt.Tick@(ib)i
*003e A8           1                                        stv.u3     0
*003f              0   ;
*003f A0           1                                        ldv.u3     0
*0040 A1           1                                        ldv.u3     1
:0041 E7 FFC2      3                                        calls.i16  T.Stmt.Tick@(ib)i
*0044 A8           1                                        stv.u3     0
*0045              0   ;
*0045 90           1                                        ldc.i3     0
*0046 A9           1                                        stv.u3     1
*0047              0   ;
*0047 A0           1                                        ldv.u3     0
*0048 A1           1                                        ldv.u3     1
:0049 E7 FFBA      3                                        calls.i16  T.Stmt.Tick@(ib)i
*004c A8           1                                        stv.u3     0
*004d              0   ;
*004d 91           1                                        ldc.i3     1
*004e A9           1                                        stv.u3     1
*004f              0   ;
*004f A0           1                                        ldv.u3     0
*0050 A1           1                                        ldv.u3     1
:0051 E7 FFB2      3                                        calls.i16  T.Stmt.Tick@(ib)i
*0054 A8           1                                        stv.u3     0
*0055              0   ;
*0055 A0           1                                        ldv.u3     0
*0056 A1           1                                        ldv.u3     1
:0057 E7 FFAC      3                                        calls.i16  T.Stmt.Tick@(ib)i
*005a A8           1                                        stv.u3     0
*005b              0   ;
 005b FF 87        2                                        trap       87                          ; putn
*005d              0   ;
*005d 03           1                                        exit
:005e   [0094]     0   T.Stmt._init@()v
*005e 04           1                                        ret
:005f   [0095]     0   $Component_End
:005f E7 FFFF      3                                        calls.i16  T.Stmt._init@()v
:0062 E7 FFC7      3                                        calls.i16  T.Stmt.Main@()v
*0065 00           1                                        halt
/0066 54 54 ..     7   T.Stmt                               .cstring   "T.Stmt"
/006d 54 54 ..    28   Test 09: if-else Statement           .cstring   "Test 09: if-else Statement"
/0089 39 39 ..    12   9|0|9|0|1|                           .cstring   "9|0|9|0|1|"

Generate 'exe' file 'T09.exe' with 149 bytes
```

2.10 Test 10: while Statement

Test Program:

```
void Main() {
    puts("Test 10: while Statement - countdown\n");
    puts("9876543210\n");

    var int sec = 9;

    while (sec >= 0)
        puti(sec--);

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                               Name        Operand                          Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0021       3                                       br.i16      $Component_End
:0003   [0003]      0   T.Stmt.Main@()v
*0003 71            1                                       enter       1
*0004               0   ;
:0004 D5 002B       3                                       lda.i16     Test 10: while Statement - countdown
 0007 FF 85         2                                       trap        85                               ; puts
*0009               0   ;
:0009 D5 004C       3                                       lda.i16     9876543210
 000c FF 85         2                                       trap        85                               ; puts
*000e               0   ;
*000e D9 09         2                                       ldc.i8      9
*0010 A8            1                                       stv.u3      0
*0011               0   ;
:0011 E0 07         2                                       br.i8       $2
:0013   [0019]      0   $3
*0013 A0            1                                       ldv.u3      0
*0014 B4 00         2                                       decv.u8     0
 0016 FF 82         2                                       trap        82                               ; puti
:0018   [0024]      0   $2
*0018 A0            1                                       ldv.u3      0
*0019 90            1                                       ldc.i3      0
*001a 1C            1                                       tlt
:001b E3 F8         2                                       brf.i8      $3
:001d   [0029]      0   $1
*001d               0   ;
 001d FF 87         2                                       trap        87                               ; putn
*001f               0   ;
*001f 03            1                                       exit
:0020   [0032]      0   T.Stmt._init@()v
*0020 04            1                                       ret
:0021   [0033]      0   $Component_End
:0021 E7 FFFF       3                                       calls.i16   T.Stmt._init@()v
:0024 E7 FFDF       3                                       calls.i16   T.Stmt.Main@()v
*0027 00            1                                       halt
/0028 54 54 ..      7   T.Stmt                              .cstring    "T.Stmt"
/002f 54 54 ..     38   Test 10: while Statement - countdow .cstring    "Test 10: while Statement - countdown"
/0055 39 39 ..     12   9876543210                          .cstring    "9876543210"

Generate 'exe' file 'T10.exe' with 97 bytes
```

2.11 Test 11: break Statement

Test Program:

```
void Main() {
    puts("Test 11: break Statement\n");
    puts("9876543210\n");

    var int sec = 9;

    while (true) {
        if (sec < 0) break;
        puti(sec--);
    }

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                                Name        Operand                          Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0025       3                                        br.i16      $Component_End
:0003   [0003]      0   T.Stmt.Main@()v
*0003 71            1                                        enter       1
*0004               0   ;
:0004 D5 002F       3                                        lda.i16     Test 11: break Statement
 0007 FF 85         2                                        trap        85                               ; puts
*0009               0   ;
:0009 D5 0044       3                                        lda.i16     9876543210
 000c FF 85         2                                        trap        85                               ; puts
*000e               0   ;
*000e D9 09         2                                        ldc.i8      9
*0010 A8            1                                        stv.u3      0
*0011               0   ;
:0011 E0 0E         2                                        br.i8       $2
:0013   [0019]      0   $3
*0013               0   ;
*0013 A0            1                                        ldv.u3      0
*0014 90            1                                        ldc.i3      0
*0015 1C            1                                        tlt
:0016 E3 04         2                                        brf.i8      $4
:0018 E0 09         2                                        br.i8       $1
:001a   [0026]      0   $4
*001a               0   ;
*001a A0            1                                        ldv.u3      0
*001b B4 00         2                                        decv.u8     0
 001d FF 82         2                                        trap        82                               ; puti
*001f               0   ;
:001f   [0031]      0   $2
:001f E0 F4         2                                        br.i8       $3
:0021   [0033]      0   $1
*0021               0   ;
 0021 FF 87         2                                        trap        87                               ; putn
*0023               0   ;
*0023 03            1                                        exit
:0024   [0036]      0   T.Stmt._init@()v
*0024 04            1                                        ret
:0025   [0037]      0   $Component_End
:0025 E7 FFFF       3                                        calls.i16   T.Stmt._init@()v
:0028 E7 FFDB       3                                        calls.i16   T.Stmt.Main@()v
*002b 00            1                                        halt
/002c 54 54 ..      7   T.Stmt                               .cstring    "T.Stmt"
/0033 54 54 ..     26   Test 11: break Statement             .cstring    "Test 11: break Statement"
/004d 39 39 ..     12   9876543210                           .cstring    "9876543210"

Generate 'exe' file 'T11.exe' with 89 bytes
```

2.12 Test 12: Bit functions

Test Program:

```
int Set(int value, int bit) {
    return value |= (1 << bit);
}

int Clear(int value, int bit) {
    return value &= ~(1 << bit);
}

int Toggle(int value, int bit) {
    return value ^= (1 << bit);
}

int Read(int value, int bit) {
    return (value >> bit) & 0x01;
}

void Main() {
    puts("Test 12: Bit functions\n");
    puts("|00000000|00000004|00000000|00000004|00000001|00000000\n");

    var int i = 0x00;
    var int r = 0x00;

    putc('|'); putx(i);
    i = Bit.Set(i, 2);
    putc('|'); putx(i);
    i = Bit.Clear(i, 2);
    putc('|'); putx(i);
    i = Bit.Toggle(i, 2);
    putc('|'); putx(i);

    r = Bit.Read(i, 2);
    putc('|'); putx(r);
    r = Bit.Read(i, 0);
    putc('|'); putx(r);

    putn();
}
```

Corresponding code generation:

```
sAddr Obj. Code    Size Label                              Name        Operand                         Comment
:0000   [0000]      0   $Component_Begin
:0000 E1 0086       3                                      br.i16      $Component_End
:0003   [0003]      0   T.Bit.Set@(ii)i
*0003 88            1                                      enter       24
*0004 A0            1                                      ldv.u3      0
*0005 91            1                                      ldc.i3      1
*0006 A1            1                                      ldv.u3      1
*0007 18            1                                      shl
*0008 0E            1                                      or
*0009 02            1                                      dup
*000a A8            1                                      stv.u3      0
*000b 03            1                                      exit
*000c 03            1                                      exit
:000d   [0013]      0   T.Bit.Clear@(ii)i
*000d 88            1                                      enter       24
*000e A0            1                                      ldv.u3      0
*000f 91            1                                      ldc.i3      1
*0010 A1            1                                      ldv.u3      1
*0011 18            1                                      shl
*0012 97            1                                      ldc.i3      -1
*0013 0F            1                                      xor
*0014 0D            1                                      and
*0015 02            1                                      dup
*0016 A8            1                                      stv.u3      0
*0017 03            1                                      exit
*0018 03            1                                      exit
:0019   [0025]      0   T.Bit.Toggle@(ii)i
*0019 88            1                                      enter       24
*001a A0            1                                      ldv.u3      0
*001b 91            1                                      ldc.i3      1
*001c A1            1                                      ldv.u3      1
*001d 18            1                                      shl
*001e 0F            1                                      xor
*001f 02            1                                      dup
*0020 A8            1                                      stv.u3      0
*0021 03            1                                      exit
*0022 03            1                                      exit
:0023   [0035]      0   T.Bit.Read@(ii)i
*0023 88            1                                      enter       24
*0024 A0            1                                      ldv.u3      0
*0025 A1            1                                      ldv.u3      1
*0026 19            1                                      shr
*0027 91            1                                      ldc.i3      1
*0028 0D            1                                      and
*0029 03            1                                      exit
*002a 03            1                                      exit
:002b   [0043]      0   T.Bit.Main@()v
*002b 72            1                                      enter       2
*002c               0   ;
:002c D5 0067       3                                      lda.i16     Test 12: Bit functions
 002f FF 85         2                                      trap        85                              ; puts
*0031               0   ;
:0031 D5 007A       3                                      lda.i16     |00000000|00000004|00000000|00000004|00000001|
 0034 FF 85         2                                      trap        85                              ; puts
*0036               0   ;
*0036 90            1                                      ldc.i3      0
*0037 A8            1                                      stv.u3      0
*0038               0   ;
*0038 90            1                                      ldc.i3      0
*0039 A9            1                                      stv.u3      1
*003a               0   ;
```

```
*003a D9 7C        2                                                  ldc.i8      124
 003c FF 81        2                                                  trap        81                        ; putc
*003e              0    ;
*003e A0           1                                                  ldv.u3      0
 003f FF 86        2                                                  trap        86                        ; putx
*0041              0    ;
*0041 A0           1                                                  ldv.u3      0
*0042 92           1                                                  ldc.i3      2
:0043 E7 FFC0      3                                                  calls.i16   T.Bit.Set@(ii)i
*0046 A8           1                                                  stv.u3      0
*0047              0    ;
*0047 D9 7C        2                                                  ldc.i8      124
 0049 FF 81        2                                                  trap        81                        ; putc
*004b              0    ;
*004b A0           1                                                  ldv.u3      0
 004c FF 86        2                                                  trap        86                        ; putx
*004e              0    ;
*004e A0           1                                                  ldv.u3      0
*004f 92           1                                                  ldc.i3      2
:0050 E7 FFBD      3                                                  calls.i16   T.Bit.Clear@(ii)i
*0053 A8           1                                                  stv.u3      0
*0054              0    ;
*0054 D9 7C        2                                                  ldc.i8      124
 0056 FF 81        2                                                  trap        81                        ; putc
*0058              0    ;
*0058 A0           1                                                  ldv.u3      0
 0059 FF 86        2                                                  trap        86                        ; putx
*005b              0    ;
*005b A0           1                                                  ldv.u3      0
*005c 92           1                                                  ldc.i3      2
:005d E7 FFBC      3                                                  calls.i16   T.Bit.Toggle@(ii)i
*0060 A8           1                                                  stv.u3      0
*0061              0    ;
*0061 D9 7C        2                                                  ldc.i8      124
 0063 FF 81        2                                                  trap        81                        ; putc
*0065              0    ;
*0065 A0           1                                                  ldv.u3      0
 0066 FF 86        2                                                  trap        86                        ; putx
*0068              0    ;
*0068 A0           1                                                  ldv.u3      0
*0069 92           1                                                  ldc.i3      2
:006a E7 FFB9      3                                                  calls.i16   T.Bit.Read@(ii)i
*006d A9           1                                                  stv.u3      1
*006e              0    ;
*006e D9 7C        2                                                  ldc.i8      124
 0070 FF 81        2                                                  trap        81                        ; putc
*0072              0    ;
*0072 A1           1                                                  ldv.u3      1
 0073 FF 86        2                                                  trap        86                        ; putx
*0075              0    ;
*0075 A0           1                                                  ldv.u3      0
*0076 90           1                                                  ldc.i3      0
:0077 E7 FFAC      3                                                  calls.i16   T.Bit.Read@(ii)i
*007a A9           1                                                  stv.u3      1
*007b              0    ;
*007b D9 7C        2                                                  ldc.i8      124
 007d FF 81        2                                                  trap        81                        ; putc
*007f              0    ;
*007f A1           1                                                  ldv.u3      1
 0080 FF 86        2                                                  trap        86                        ; putx
*0082              0    ;
 0082 FF 87        2                                                  trap        87                        ; putn
*0084              0    ;
*0084 03           1                                                  exit
```

```
:0085   [0133]      0   T.Bit._init@()v
*0085 04            1                                       ret
:0086   [0134]      0   $Component_End
:0086 E7 FFFF       3                                       calls.i16   T.Bit._init@()v
:0089 E7 FFA2       3                                       calls.i16   T.Bit.Main@()v
*008c 00            1                                       halt
/008d 54 54 ..      6   T.Bit                               .cstring    "T.Bit"
/0093 54 54 ..     24   Test 12: Bit functions              .cstring    "Test 12: Bit functions"
/00ab 7C 7C ..     56   |00000000|00000004|00000000|00000004|00000001|00000000
                                                            .cstring    "|00000000|00000004|00000000|00000004|00000001

Generate 'exe' file 'T12.exe' with 227 bytes
```

Index