Features and restrictions of Oberon-ARM

N. Wrth, 6.12.2007, rev. 17.12.2007

This memo describes the restrictions and extensions of Oberon recently implemented for the ARM processor with respect to Revised Oberon (Oberon-07).

1. Restrictions

The type LONGREAL does not exist.

Neither do LONGINT and SHORTINT (not either in Oberon-07).

Open arrays (procedure parameters) cannot be multi-dimensional (matrices).

The increment *n* of the standard procedures INC and DEC must be in the range 0 <= n < 256.

The step *n* in for clauses must be in the range -256 < n < 256.

At most 256 cases are admitted in case statements.

Note that export/import of variables is not allowed, except in read-only mode for variables of a scalar type.

2. Assertions

Assertions were introduced in Revised Oberon by the standard procedure ASSERT(b, n). If n is omitted, its default is 0. In Oberon-ARM the short form *!b* may be used in place of ASSERT(b).

3. Open Arrays

In Standard Oberon open arrays, often also called dynamic arrays, can occur only as formal parameters. In Oberon-ARM this facility has been extended to variables. In this case, the length is determined "dynamically", i.e. during program execution by a statement NEW. Let a variable *a* be declared as

```
VAR a: ARRAY OF INTEGER                    (length is not specified)
```

Then a statement NEW(a, n) must be executed before accessing *a*, where *n* is an expression defining the length of the array *a*. Open arrays cannot be components of records or arrays.

4. Leaf Procedures

If a procedure is not calling any other procedure, it is called a *leaf procedure*. This may be indicated by following the symbol PROCEDURE by an asterisk. It serves as a compiler directive, and it causes the compiler to let parameters remain allocated in registers (of which there are at most 10 available). Furthermore, the variables declared first in the declaration section are also allocated in registers, if possible. Only variables of type INTEGER or SET are considered. This facility serves to speed up execution.

5. Interrupt Handlers

Interrupt procedures are commands that are activated by an interrupt signal. They have neither parameters nor a result. They are denoted by an integer enclosed in brackets in place of a parameter list. The integer specifies the offset in the return instruction. This is a peculiar feature of the ARM processor. An asterisk after the symbol PROCEDURE has the additional effect that no registers are saved. This option is used in the case of the "fast interrupt" (FIQ), where the ARM-processor uses a different set of registers R8 – R15.

```
PROCEDURE IRQ [4];
BEGIN INC(intcount)  (*intcount is a global variable for counting interrupts*)
END IRQ;
```

The handler procedure must be assigned to the corresponding interrupt location. This must occur in the body of the module in which the handler is declared. It is done by using the procedure SYSTEM.PUT. Also, the respective interrupt must be enabled (see SYSTEM.LDPSR below).

Interrupt source	interrupt location	offset integer
UND	4	0
SWI	8	0
ABT	16	8
IRQ	24	4
FIQ	28	4

6. Low-level Facilities in Module SYSTEM

The module SYSTEM contains definitions that are necessary to program low-level operations referring directly to resources particular to a given computer and/or implementation. These include for example facilities for accessing devices that are controlled by the computer, and perhaps facilities to break the data type compatibility rules otherwise imposed by the language definition. It is recommended to restrict their use to specific low-level modules, as such modules are inherently non-portable and not “type-safe”. However, they are easily recognized due to the identifier SYSTEM appearing in their import lists. The subsequent definitions are specifically available in the Oberon implemenation for the ARM architecture.

The procedures contained in module SYSTEM are listed in the following tables. They correspond to short instruction sequences compiled as in-line code. v stands for a variable, x, y, a, and n for expressions.

Function procedures:

Name	Argument types	Result type	Function
ADR(v)	any	INTEGER	address of variable v
SIZE(T)	any type	INTEGER	size in bytes
BIT(a, n)	a, n: INTEGER	BOOLEAN	bit n of mem[a]
ROR(x, n)	x: INTEGER	type of x	rotate right
VAL(T, x)	T, x: any type	T	x cast into type T

Proper procedures:

Name	Argument types	Function
GET(a, v)	a: INTEGER; v: any basic type	v := mem[a]
PUT(a, x)	a: INTEGER; x: any basic type	mem[a] := x
LDPSR(b, x)	b, x: INTEGER	load status register with x
STPSR(b, v)	b, v: INTEGER	store status register in v
LDCPR(p, n, x)	p, n, x: INTEGER	load register n of coprocessor p with x
STCPR(p, n, v)	p, n, v: INTEGER	load register n of coprocessor p in v

Module SYSTEM also exports the data type BYTE. No representation of values is specified. It is used to relax the type compatibility rules for procedure parameters. If a formal parameter is of type ARRAY OF SYSTEM.BYTE, then the corresponding actual parameter may be of any type. In addition, the type BYTE is assignment compatible with CHAR.

The procedures PUT and GET, and the Boolean function BIT, are used to access device registers in memory mapped interfaces. Their first parameter is the address of the register (memory cell) to be accessed.

PUT(a, x)	mem[a] := x
GET(a, v)	v := mem[a]
BIT(a, n)	bit n of mem[a]

Whereas PUT, GET, and BIT are considered as standard facilities of Oberon systems, the following procedures are ARM-specific. The ARM processor contains some special registers requiring special access procedures. The *program status register* is set by procedure LDPSR and read by STPSR, the coprocessor registers are loaded by LDCPR and read by STCPT.

Load Program Status Register

SYSTEM.LDPSR(u, src) MSR

u = 0 PSR of current processor mode
u = 1 PSR of saved processor mode
src value to be loaded in PSR, an expression

Store Program Status Register

SYSTEM.STPSR(u, dst) MRS

u = 0 PSR of current processor mode
u = 1 PSR of saved processor mode

The following are examples of accessing the PSR. They switch to a new mode:

```
SYSTEM.LDPSR(1, 0D1H)       to FIQ mode
SYSTEM.LDPSR(1, 0D2H)       to IRQ mode
SYSTEM.LDPSR(1, 0D3H)       to SVC mode
SYSTEM.LDPSR(1, 10H)  to user mode, enable interrupts
```

Load Coprocessor Register

SYSTEM.LDCPR(code, dst, src) MCR

code a hex constant of the form a00bcdH, where
  a opcode1 * 2
  b coprocessor number (0 … 15)
  c opcode2 * 2
  d coprocessor register2 number
dst coprocessor register1 number
src value to be loaded into dst, an expression

Store Coprocessor Register

SYSTEM.STCPR(code, src, dst) MRC

code as for LDCPR
src coprocessor register1 number
dst variable to which value is to be assigned

Flush cashes

```
SYSTEM.FLUSH(17H)   flush I-cache and D-cache
SYSTEM.FLUSH(9AH)   drain write buffer
```