Current problem:

Special registers class have their CheckPermission() rule -> need many special registers class and they are only different in CheckPermission() function.

Try to improve:
Remove these classes of small difference.

EP 2199:

D7.3.19 MDRAR_EL1, Monitor Debug ROM Address Register

The MDRAR_EL1 characteristics are:

Purpose

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system.

Usage constraints

This register is accessible as follows:

EL0	EL1 (NS)	EL1 (S)	EL2 (NS)	EL3 (SCR.NS=1)	EL3 (SCR.NS=0)
-	RO	RO	RO	RO	RO

Traps and Enables

For a description of the prioritization of any exceptions, see *Synchronous exception prioritization* on page D1-1547.

If MDCR_EL2.TDRA==1, Non-secure accesses to this register will trap from EL1 and EL0 to EL2.

If MDCR_EL3.TDA==1, accesses to this register will trap from EL2, EL1 and EL0 to EL3.

__define_sys_reg_with_opcode(ClsSysReg, 32, MDRAR, EL1, "MDRAR_EL1", ...)
Define_Trap("1==QRegField(MDCR,EL2,TDRA) && NSAccess() && level <= EL1", EL2)
Define_Trap("1==QRegField(MDCR,EL3,TDA) && level <= EL2", EL3)
Define_EL_MODE(E1.NS=RO, E1.S=RO, ...)

__define_sys_reg_with_opcode(ClsSysReg, 32, MDRAR, EL1, "MDRAR_EL1", ...)
Define_Trap("1==MDCR_EL2.TDRA && NS && level <= EL1", EL2)
Define_Trap("1==MDCR_EL3.TDA && level <= EL2", EL3)
...

CodeGen:

CheckWritePermission_MDRAR_EL1(level, el[4], ...)
{
 If (1==QRegField(MDCR,EL2,TDRA) && NSAccess() && level <= EL1) return EL2;
 If (level==1 && RO==el[1].mode && SECURE==e1[1].type) return EL2;
 If (level==1 && RO==el[1].mode && NON_SECURE==e1[1].type) return EL2;
 If (level==2 && RO==el[2].mode && SECURE==e1[1].type) return EL3;
 ...
 TypUlong64 tdra;

 tdra = context.aarch64.pSysRegManager->QueryReg(EL2, MDCR)->ReadByField(nsFIELD::TDRA, tdra);

```
 If (1==tdra && NSAccess() && level <= EL1) return EL2;

 TypUlong64 tda;
 Tdra = context.aarch64.pSysRegManager->QueryReg(EL3, MDCR)-
>ReadByField(nsFIELD::TDA, tda);

 If (1==tda && level <= EL2) return EL3;

 return NO_TRAP;
}


Test:
ClsSysReg.CheckTrap = CheckTrap_MDRAR_EL1;   // set function
pointer
Cls.SysReg->CheckTrap(level);    // trigger test
```

Solution:

EP 2199:

D7.3.19 MDRAR_EL1, Monitor Debug ROM Address Register

The MDRAR_EL1 characteristics are:

Purpose

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system.

Usage constraints

This register is accessible as follows:

EL0	EL1 (NS)	EL1 (S)	EL2 (NS)	EL3 (SCR.NS=1)	EL3 (SCR.NS=0)
-	RO	RO	RO	RO	RO

Traps and Enables

For a description of the prioritization of any exceptions, see *Synchronous exception prioritization* on page D1-1547.

If MDCR_EL2.TDRA==1, Non-secure accesses to this register will trap from EL1 and EL0 to EL2.

If MDCR_EL3.TDA==1, accesses to this register will trap from EL2, EL1 and EL0 to EL3.

__define_sys_reg_with_opcode(ClsSysReg, 32, MDRAR, EL1, "MDRAR_EL1", ...)
Define_Trap("1==MDCR_EL2.TDRA && NS && level <= EL1", EL2)
Define_Trap("1==MDCR_EL3.TDA && level <= EL2", EL3)
...

CodeGen:

```
CheckPermission_MDRAR_EL1(int RW, TypExceptionLevel level, int op0)
{
  TypException ret = ClsRTLSyncSysReg::CheckReadPermission(level, context);
  if(NO_TRAP == ret)
  {
    If (1==QueryRegField(MDCR,EL2,TDRA) && !IsSecureAccess() && level <= EL1) return Trap(EL2);
    If (1==QueryRegField(MDCR,EL3,TDA) && level <= EL2) return Trap(EL3);
  }
  return ret;
}
```

Test:

```
ClsSysReg.CheckPermissionPtr = CheckPermission_MDRAR_EL1; // set function pointer
Cls.SysReg->CheckPermission(level);    // trigger test
```

P 2221:

D7.4.2 PMCCNTR_EL0, Performance Monitors Cycle Count Register

EL0 EL1 (NS) EL1 (S) EL2 (NS) EL3 (SCR.NS=1) EL3 (SCR.NS=0)
Config-RW RO RO RW RW RW

If PMUSERENR_EL0.CR==0, and PMUSERENR_EL0.EN==0, read accesses to this register will trap from EL0 to EL1.

__define_sys_reg_with_opcode(ClsSysReg, 32, PMCCNTR, EL0, "PMCCNTR _EL0", ...)
Define_Trap(“level==EL1 && WRITE”,undefine)
Define_Trap(“0==PMUSERENR_EL0.CR && 0==PMUSERENR_EL0.EN && level == EL0”, EL1,opcode)

CodeGen:

```
CheckPermission_ PMCCNTR _EL0(int RW, TypExceptionLevel level, int op0)
{
  TypException ret = ClsRTLSyncSysReg::CheckReadPermission(level, context);
  if(NO_TRAP == ret)
  {
     If (level==EL1 && WRITE==RW) return Trap(UNDEF, op0);
     If (READ=RW && 0== QRegField(PMUSERENR,EL0,CR)==0 &&
      QRegField(PMUSERENR,EL0,EN)==0 && level == EL0) return
      Trap(EL1, op0);
  }
  return ret;
}
```

```
TypException Trap(TypExceptionLevel el, int op0)
{
  TypWord opcode;
  FaultRecord fault;
            context.aarch64.pMMU-
   >ReadInsn(context.aarch64.pProcessorState->ReadPC(), opcode,
   context, fault);

  context.aarch64.pExceptionManager->SystemRegisterTrap(el, op0,
opcode, context);

  return el;
}

Int QRegField(PMUSERENR,EL0,CR)
{
}

bool IsSecureAccess()
{
}
```