CPYFPRTRN, CPYFMRTRN, CPYFERTRN

Memory Copy Forward-only, reads unprivileged and non-temporal. These instructions perform a memory copy. The prologue, main, and epilogue instructions are expected to be run in succession and to appear consecutively in memory: CPYFPRTRN, then CPYFMRTRN, and then CPYFERTRN.

CPYFPRTRN performs some preconditioning of the arguments suitable for using the CPYFMRTRN instruction, and performs an implementation defined amount of the memory copy. CPYFMRTRN performs an implementation defined amount of the memory copy. CPYFERTRN performs the last part of the memory copy.

Note

The inclusion of implementation defined amounts of memory copy allows some optimization of the size that can be performed.

The memory copy performed by these instructions is in the forward direction only, so the instructions are suitable for a memory copy only where there is no overlap between the source and destination locations, or where the source address is greater than the destination address.

The architecture supports two algorithms for the memory copy: option A and option B. Which algorithm is used is implementation defined.

Note

Portable software should not assume that the choice of algorithm is constant.

After execution of CPYFPRTRN, option A (which results in encoding PSTATE.C = 0):

- If Xn<63> == 1, the copy size is saturated to 0x7FFFFFFFFFFFFFFF.
- Xs holds the original Xs + saturated Xn.
- Xd holds the original Xd + saturated Xn.
- Xn holds -1* saturated Xn + an implementation defined number of bytes copied.
- PSTATE.{N,Z,V} are set to {0,0,0}.

After execution of CPYFPRTRN, option B (which results in encoding PSTATE.C = 1):

- If Xn<63> == 1, the copy size is saturated to 0x7FFFFFFFFFFFFFFF.

- Xs holds the original Xs + an implementation defined number of bytes copied.
- Xd holds the original Xd + an implementation defined number of bytes copied.
- Xn holds the saturated Xn - an implementation defined number of bytes copied.
- PSTATE.{N,Z,V} are set to {0,0,0}.

For CPYFMRTRN, option A (encoded by PSTATE.C = 0), the format of the arguments is:

- Xn is treated as a signed 64-bit number and holds -1* the number of bytes remaining to be copied in the memory copy in total.
- Xs holds the lowest address that the copy is copied from -Xn.
- Xd holds the lowest address that the copy is made to -Xn.
- At the end of the instruction, the value of Xn is written back with -1* the number of bytes remaining to be copied in the memory copy in total.

For CPYFMRTRN, option B (encoded by PSTATE.C = 1), the format of the arguments is:

- Xn holds the number of bytes remaining to be copied in the memory copy in total.
- Xs holds the lowest address that the copy is copied from.
- Xd holds the lowest address that the copy is copied to.
- At the end of the instruction:
    - the value of Xn is written back with the number of bytes remaining to be copied in the memory copy in total.
    - the value of Xs is written back with the lowest address that has not been copied from.
    - the value of Xd is written back with the lowest address that has not been copied to.

For CPYFERTRN, option A (encoded by PSTATE.C = 0), the format of the arguments is:

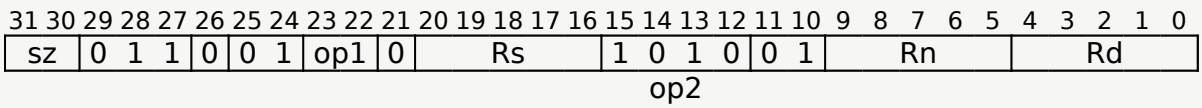
- Xn is treated as a signed 64-bit number and holds -1* the number of bytes remaining to be copied in the memory copy in total.
- Xs holds the lowest address that the copy is copied from -Xn.
- Xd holds the lowest address that the copy is made to -Xn.
- At the end of the instruction, the value of Xn is written back with 0.

For CPYFERTRN option B (encoded by PSTATE.C = 1), the format of the arguments is:

- Xn holds the number of bytes remaining to be copied in the memory copy in total.
- Xs holds the lowest address that the copy is copied from.
- Xd holds the lowest address that the copy is copied to.
- At the end of the instruction:
    - the value of Xn is written back with 0.

- the value of Xs is written back with the lowest address that has not been copied from.
- the value of Xd is written back with the lowest address that has not been copied to.

Integer
(FEAT_MOPS)



Epilogue (op1 == 10)

```
CPYFERTRN [<Xd>]!, [<Xs>]!, <Xn>!
```

Main (op1 == 01)

```
CPYFMRTRN [<Xd>]!, [<Xs>]!, <Xn>!
```

Prologue (op1 == 00)

```
CPYFPRTRN [<Xd>]!, [<Xs>]!, <Xn>!
```

```
if !IsFeatureImplemented(FEAT_MOPS) || sz != '00' then UNDEFINED;

integer d = UInt(Rd);
integer s = UInt(Rs);
integer n = UInt(Rn);
bits(4) options = op2;
boolean rnontemporal = options<3> == '1';
boolean wnontemporal = options<2> == '1';

MOPSStage stage;
case op1 of
    when '00' stage = MOPSStage_Prologue;
    when '01' stage = MOPSStage_Main;
    when '10' stage = MOPSStage_Epilogue;
    otherwise SEE "Memory Copy and Memory Set";

CheckMOPSEnabled();

if s == n || s == d || n == d || d == 31 || s == 31 || n == 31 then
    Constraint c = ConstrainUnpredictable(Unpredictable_MOPSOVERLAP31);
    assert c IN {Constraint_UNDEF, Constraint_NOP};
    case c of
        when Constraint_UNDEF UNDEFINED;
        when Constraint_NOP   EndOfInstruction();
```

For information about the constrained unpredictable behavior of this instruction, see *Architectural Constraints on UNPREDICTABLE behaviors*, and particularly *Memory Copy and Memory Set CPY**.

Assembler Symbols

<Xd>	For the epilogue and main variant: is the 64-bit name of the general-purpose register that holds an encoding of the destination address, encoded in the "Rd" field. For the prologue variant: is the 64-bit name of the general-purpose register that holds the destination address and is updated by the instruction, encoded in the "Rd" field.
<Xs>	For the epilogue and main variant: is the 64-bit name of the general-purpose register that holds an encoding of the source address, encoded in the "Rs" field. For the prologue variant: is the 64-bit name of the general-purpose register that holds the source address and is updated by the instruction, encoded in the "Rs" field.
<Xn>	For the epilogue variant: is the 64-bit name of the general-purpose register that holds an encoding of the number of bytes to be transferred and is set to zero at the end of the instruction, encoded in the "Rn" field. For the main variant: is the 64-bit name of the general-purpose register that holds an encoding of the number of bytes to be transferred, encoded in the "Rn" field. For the prologue variant: is the 64-bit name of the general-purpose register that holds the number of bytes to be transferred and is updated by the instruction to encode the remaining size and destination, encoded in the "Rn" field.

Operation

```
constant integer N = MaxBlockSizeCopiedBytes();
bits(64) toaddress = X[d, 64];
bits(64) fromaddress = X[s, 64];
bits(64) cpysize = X[n, 64];
bits(4) nzcv = PSTATE.<N,Z,C,V>;
bits(8*N) readdata;

boolean implements_option_a = CPYFOptionA();
boolean rprivileged = if options<1> == '1' then AArch64.IsUnprivAccessP
boolean wprivileged = if options<0> == '1' then AArch64.IsUnprivAccessP

AccessDescriptor raccdesc = CreateAccDescMOPS(MemOp_LOAD, rprivileged,
AccessDescriptor waccdesc = CreateAccDescMOPS(MemOp_STORE, wprivileged,

if stage == MOPSStage_Prologue then
    if cpysize<63> == '1' then cpysize = 0x7FFFFFFFFFFFFFFF<63:0>;

    if implements_option_a then
        nzcv = '0000';
        // Copy in the forward direction offsets the arguments.
        toaddress = toaddress + cpysize;
        fromaddress = fromaddress + cpysize;
        cpysize = Zeros(64) - cpysize;
```

```
    else
        nzcv = '0010';

else
    CheckMemCpyParams(stage, implements_option_a, nzcv, options, d, s,

bits(64) stagecpysize = MemCpyStageSize(stage, toaddress, fromaddress,

if implements_option_a then
    while SInt(stagecpysize) != 0 do
        // IMP DEF selection of the block size that is worked on. While
        // implementations might make this constant, that is not assume
        constant integer B = CPYSizeChoice(toaddress, fromaddress, cpys
        assert B <= -1 * SInt(stagecpysize);

        readdata<B*8-1:0> = Mem[fromaddress+cpysize, B, raccdesc];
        Mem[toaddress+cpysize, B, waccdesc] = readdata<B*8-1:0>;

        cpysize = cpysize + B;
        stagecpysize = stagecpysize + B;

        if stage != MOPSStage_Prologue then
            X[n, 64] = cpysize;
else
    while UInt(stagecpysize) > 0 do
        // IMP DEF selection of the block size that is worked on. While
        // implementations might make this constant, that is not assume
        constant integer B = CPYSizeChoice(toaddress, fromaddress, cpys
        assert B <= UInt(stagecpysize);

        readdata<B*8-1:0> = Mem[fromaddress, B, raccdesc];
        Mem[toaddress, B, waccdesc] = readdata<B*8-1:0>;

        fromaddress = fromaddress + B;
        toaddress = toaddress + B;
        cpysize = cpysize - B;
        stagecpysize = stagecpysize - B;

        if stage != MOPSStage_Prologue then
            X[n, 64] = cpysize;
            X[d, 64] = toaddress;
            X[s, 64] = fromaddress;

if stage == MOPSStage_Prologue then
    X[n, 64] = cpysize;
    X[d, 64] = toaddress;
    X[s, 64] = fromaddress;
    PSTATE.<N,Z,C,V> = nzcv;
```


Internal version only: isa v33.64, AdvSIMD v29.12, pseudocode no_diffs_2023_09_RC2, sve v2023-06_rel ; Build timestamp: 2023-09-18T17:56