BTI

Branch Target Identification. A BTI instruction is used to guard against the execution of instructions which are not the intended target of a branch. Outside of a guarded memory region, a BTI instruction executes as a NOP. Within a guarded memory region while *PSTATE*.BTYPE != 0b00, a BTI instruction compatible with the current value of PSTATE.BTYPE will not generate a Branch Target Exception and will allow execution of subsequent instructions within the memory region.
The operand <targets> passed to a BTI instruction determines the values of *PSTATE*.BTYPE which the BTI instruction is compatible with.

Note

Within a guarded memory region, when *PSTATE*.BTYPE != 0b00, all instructions will generate a Branch Target Exception, other than BRK, BTI, HLT, PACIASP, and PACIBSP, which might not. See the individual instructions for more information.

System
(FEAT_BTI)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	0	1	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	x	x	0	1	1	1	1	1
																				CRm				op2							

BTI {<targets>}

```
SystemHintOp op;

if CRm:op2 == '0100 xx0' then
    op = SystemHintOp_BTI;
    // Check branch target compatibility between BTI instruction and PS
    SetBTypeCompatible(BTypeCompatible_BTI(op2<2:1>));
else
    EndOfInstruction();
```

Assembler Symbols

<targets>
Is the type of indirection, encoded in “op2<2:1>”:

op2<2:1>	<targets>
00	(omitted)
01	c
10	j
11	jc

Operation

```
case op of
    when SystemHintOp_YIELD
        Hint_Yield();

    when SystemHintOp_DGH
        Hint_DGH();

    when SystemHintOp_WFE
        integer localtimeout = 1 << 64;     // No local timeout event is
        Hint_WFE(localtimeout, WFxType_WFE);

    when SystemHintOp_WFI
        integer localtimeout = 1 << 64;     // No local timeout event is
        Hint_WFI(localtimeout, WFxType_WFI);

    when SystemHintOp_SEV
        SendEvent();

    when SystemHintOp_SEVL
        SendEventLocal();

    when SystemHintOp_ESB
        if IsFeatureImplemented(FEAT_TME) && TSTATE.depth > 0 then
            FailTransaction(TMFailure_ERR, FALSE);
        SynchronizeErrors();
        AArch64.ESBOperation();
        if PSTATE.EL IN {EL0, EL1} && EL2Enabled() then AArch64.vESBOpe
        TakeUnmaskedSErrorInterrupts();

    when SystemHintOp_PSB
        ProfilingSynchronizationBarrier();

    when SystemHintOp_TSB
        TraceSynchronizationBarrier();

    when SystemHintOp_GCSB
        GCSSynchronizationBarrier();

    when SystemHintOp_CHKFEAT
        X[16, 64] = AArch64.ChkFeat(X[16, 64]);

    when SystemHintOp_CSDB
        ConsumptionOfSpeculativeDataBarrier();

    when SystemHintOp_CLRBHB
        Hint_CLRBHB();

    when SystemHintOp_BTI
        SetBTypeNext('00');

    when SystemHintOp_NOP
        return;     // do nothing

    otherwise
        Unreachable();
```

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