**Draft Text Specification**

The proposed text changes are based on the document JCTVC-R1005-v3.doc.

The changes are marked in yellow for PU-symIBC.

The changes are marked in green for segIBC.

**Syntax**

#### Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize ) { | Descriptor |
| if( transquant\_bypass\_enabled\_flag ) |  |
| **cu\_transquant\_bypass\_flag** | ae(v) |
| if( slice\_type != I ) |  |
| **cu\_skip\_flag**[ x0 ][ y0 ] | ae(v) |
| nCbS = ( 1  <<  log2CbSize ) |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| ~~if( intra\_block\_copy\_enabled\_flag )~~ |  |
| **~~intra\_bc\_flag~~**~~[ x0 ][ y0 ]~~ | ~~ae(v)~~ |
| if( slice\_type != I ~~&& !intra\_bc\_flag[ x0 ][ y0 ]~~ | | intra\_block\_copy\_enabled\_flag ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( palette\_enabled\_flag && ChromaArrayType = = 3 &&   CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ~~&& !intra\_bc\_flag[ x0 ][ y0 ]~~) |  |
| **palette\_mode\_flag**[ x0 ][ y0 ] | ae(v) |
| if( palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| palette\_coding( x0, y0, nCbS ) |  |
| else { |  |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA | | ~~intra\_bc\_flag[ x0 ][ y0 ] | |~~  log2CbSize = = MinCbLog2SizeY ) |  |
| **part\_mode** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ~~&& !intra\_bc\_flag[ x0 ][ y0 ]~~) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag &&   log2CbSize >= Log2MinIpcmCbSizeY &&  log2CbSize <= Log2MaxIpcmCbSizeY ) |  |
| … |  |
|  |  |
| } else { /\* PART\_NxN \*/ |  |
| prediction\_unit( x0, y0, nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0, y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| prediction\_unit( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 ) |  |
| } |  |
| } |  |
| if (intra\_bc\_flag[ x0 ][ y0 ] && intra\_bc\_flag[ x0 + nCbS – 1 ][ y0 ] && intra\_bc\_flag[ x0 ][ y0 + nCbS – 1 ] && intra\_bc\_flag[ x0 + nCbS – 1 ][ y0 + nCbS – 1 ]){ |  |
| segmental\_ibc\_flag[ x0 ][ y0 ] | ae(v) |
| if(segment\_ibc\_flag[ x0 ][ y0 ]){ |  |
| for( cIdx=0; cIdx<( ChromaArrayType = = 0 ? 1 : 3); cIdx++ ) |  |
| for( j = 0; j < 2; j++ ){ |  |
| abs\_segment\_offset[ x0 ][ y0 ][ j ][ cIdx ] | ae(v) |
| if(abs\_segment\_offset[ x0 ][ y0 ][ j ][ cIdx ]>0) |  |
| segment\_offset\_sign\_flag[ x0 ][ y0 ][ j ][ cIdx ]  } | ae(v) |
| } |  |
| } |  |
| if( !pcm\_flag[ x0 ][ y0 ] && !segmental\_ibc\_flag[ x0 ][ y0 ]) { |  |
| if( ( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA &&   !( PartMode = = PART\_2Nx2N && merge\_flag[ x0 ][ y0 ] ) ) | |   ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA && intra\_bc\_flag[ x0 ][ y0 ] ) ) |  |
| rqt\_root\_cbf | ae(v) |
| if( rqt\_root\_cbf ) { |  |
| if( residual\_adaptive\_colour\_transform\_enabled\_flag &&   ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTER | | intra\_bc\_flag[ x0 ][ y0 ] | |   intra\_chroma\_pred\_mode[ x0 ][ y0 ] = = 4 ) ) |  |
| cu\_residual\_act\_flag |  |
| MaxTrafoDepth = ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ?   ( max\_transform\_hierarchy\_depth\_intra + IntraSplitFlag ) :   max\_transform\_hierarchy\_depth\_inter ) |  |
| transform\_tree( x0, y0, x0, y0, log2CbSize, 0, 0 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

#### 7.3.8.6 Prediction unit syntax

|  |  |
| --- | --- |
| prediction\_unit( x0, y0, nPbW, nPbH ) { | Descriptor |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) { |  |
| if( MaxNumMergeCand > 1 ) |  |
| **merge\_idx**[ x0 ][ y0 ] | ae(v) |
| } else {~~if( intra\_bc\_flag[ x0 ][ y0 ] ) /\* Intra BC\*/~~ |  |
| if( intra\_block\_copy\_enabled\_flag && slice\_type != I && ( PartMode < PART\_NxN  | | MinCbLog2SizeY > 3 && PartMode = = PART\_NxN ))) |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| if( intra\_bc\_flag[ x0 ][ y0 ] ) { /\* Intra BC\*/ |  |
| **flipping\_ibc\_flag**[ x0 ][ y0 ] | ae(v) |
| if(flipping\_ibc\_flag[ x0 ][ y0 ]){ |  |
| **flipping\_ibc\_direction**[ x0 ][ y0 ] | ae(v) |
| } |  |
| bvd\_coding( x0, y0, 2 ) | ae(v) |
| **bvp\_flag**[ x0 ][ y0 ] |  |
| } else { /\* MODE\_INTER \*/ |  |
| **merge\_flag**[ x0 ][ y0 ] | ae(v) |
| if( merge\_flag[ x0 ][ y0 ] ) { |  |
| if( MaxNumMergeCand > 1 ) | ae(v) |
| **merge\_idx**[ x0 ][ y0 ] |  |
| } else { |  |
| if( slice\_type = = B ) | ae(v) |
| **inter\_pred\_idc**[ x0 ][ y0 ] |  |
| if( inter\_pred\_idc[ x0 ][ y0 ] != PRED\_L1 ) { |  |
| if( num\_ref\_idx\_l0\_active\_minus1 > 0 ) | ae(v) |
| **ref\_idx\_l0**[ x0 ][ y0 ] |  |
| mvd\_coding( x0, y0, 0 ) |  |
| **mvp\_l0\_flag**[ x0 ][ y0 ] | ae(v) |
| } |  |
| if( inter\_pred\_idc[ x0 ][ y0 ] != PRED\_L0 ) { | ae(v) |
| if( num\_ref\_idx\_l1\_active\_minus1 > 0 ) |  |
| **ref\_idx\_l1**[ x0 ][ y0 ] |  |
| if( mvd\_l1\_zero\_flag &&   inter\_pred\_idc[ x0 ][ y0 ] = = PRED\_BI ) { |  |
| MvdL1[ x0 ][ y0 ][ 0 ] = 0 | ae(v) |
| MvdL1[ x0 ][ y0 ][ 1 ] = 0 |  |
| } else |  |
| mvd\_coding( x0, y0, 1 ) |  |
| **mvp\_l1\_flag**[ x0 ][ y0 ] |  |
| } |  |
| } | ae(v) |
| } |  |
| } |  |
| } |  |
|  |  |
|  |  |

**Semantics**

#### Coding unit semantics

**~~intra\_bc\_flag~~**~~[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded in intra block copying mode. intra\_bc\_flag[ x0 ][ y0 ] equal to 0 specifies that the current coding unit is coded according to pred\_mode\_flag. When not present, the value of intra\_bc\_flag is inferred to be equal to 0. The array indices x0 and y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.~~

**segmental\_ibc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the considered coding unit is coded in segmental intra block copy mode. segmental\_ibc\_flag [ x0 ][ y0 ] equal to 0 specifies that the considered coding unit is not coded in segmental intra block copying mode. When not present, the value of segmental\_ibc\_flag is inferred to be equal to 0. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

**abs\_segment\_offset**[ x0 ][ y0 ][ j ][ cIdx ] specifies the absolute value of the segment offset for the jth segment in the considered coding unit. When not present, the value of abs\_segment\_offset[ x0 ][ y0 ][ j ][ cIdx ] is inferred to be equal to 0.. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for luma, equal to 1 for Cb, and equal to 2 for Cr.

**segment\_offset\_sign**[ x0 ][ y0 ][ j ][ cIdx ] specifies the sign of the offset for the jth segment in the considered coding unit. When not present, the value of segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ] is inferred to be equal to 0. If segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ] is equal to 0, the value of the segment offset for the jth segment in the considered coding unit is positive.Otherwise (segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ] is equal to 1), the segment offset for the jth segment in the considered coding unit is negative. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for luma, equal to 1 for Cb, and equal to 2 for Cr.

The variable SegOffset[ x0 ][ y0 ][ j ][ cIdx ] for j = 0, … nSegNum[ x0 ][ y0 ], cIdx = 0,… ( ChromaArrayType = = 0 ? 1 : 3) is derived as follows:

SegOffset[ x0 ][ y0 ][ j ][ cIdx ]=abs\_segment\_offset[ x0 ][ y0 ][ j ][ cIdx ]\*(1-2\*segment\_offset\_sign[ x0 ][ y0 ][ j ][ cIdx ]).

…

#### Prediction unit semantics

**intra\_bc\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current prediction unit is coded in intra block copying mode. intra\_bc\_flag[ x0 ][ y0 ] equal to 0 specifies that the current prediction unit is coded according to pred\_mode\_flag. The array indices x0 and y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture. When not present, the value of intra\_bc\_flag is inferred as follows:

* + if pred\_mode\_flag is equal to 0 and intra\_block\_copy\_enabled\_flag is equal to 1
    - if slice\_type is equal to I, intra\_bc\_flag is set equal to 1;
    - if PartMode is equal to PART\_NxN and MinCbLog2SizeY is equal to 3, intra\_bc\_flag is set equal to 1;
  + otherwise, intra\_bc\_flag is set equal to 0

**flipping\_ibc\_flag**[ x0 ][ y0] equals to 1 specifies that the current prediction unit is coded with flipped intra block copy mode. If the flipping\_ibc\_flag[ x0 ][ y0 ] equals to 0, it specifies that the current prediction unit is using normal intra block copy mode. The array indices x0 and y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture. When not present, the value of flipping\_ibc\_flag is inferred to be equal to 0.

**flipping\_ibc\_direction\_**[ x0 ][ y0 ] equal to 1 specifies that the current flipping direction is vertical. If the **flipping\_ibc\_direction**[ x0 ][ y0 ] equal to 0 specifies that the current flipping direction is horizontal. The array indices x0 and y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

**Decoding process**

## Decoding process for coding units coded in intra prediction mode

## 8.4.1 General decoding process for coding units coded in intra prediction mode

……

### Decoding process for intra blocks

#### General decoding process for intra blocks

### ……

##### Specification of intra block copying prediction mode

The (nTbS)x(nTbS) array of predicted samples samples, with x, y = 0..nTbS − 1, is derived as follows:

– The reference sample location (xRefCmp, yRefCmp ) is specified by:

( xRefCmp, yRefCmp ) = ( xTbCmp + x + bv[ 0 ], yTbCmp + y + bv[ 1 ] ) (‑)

* If flipping\_ibc\_flag[ xTbY ][  yTbY ] is equal to 0. each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ y ];
* Otherwise (flipping\_ibc\_flag[ xTbY ][  yTbY ] is equal to 1),
* if flipping\_ibc\_direction [ xTbY ][ yTbY ] is equal to 0, each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ nTbS-1-x ][ y ];
* Otherwise (flipping\_ibc\_direction [ xTbY ][ yTbY ] is equal to 1), each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ nTbS-1-y ].

…

The (nTbS)x(nTbS) array of predicted samples samples, with x, y = 0..nTbS − 1, is derived as follows:

– The reference sample location (xRefCmp, yRefCmp ) is specified by:

( xRefCmp, yRefCmp ) = ( xTbCmp + x + bv[ 0 ], yTbCmp + y + bv[ 1 ] ) (‑)

Each sample at the location ( xRefCmp, yRefCmp ) is assigned to predSamples[ x ][ y ].

When segmental\_ibc\_flag[ xTb0 ][ yTb0 ] is equal to 1, the following applies in order:

1. A variable predAvg is derived as follows:

predAvg =,

where k = Log2( nTbS\*nTbS ).

1. A variable predMin is set equal to the minimum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1. A variable predMax is set equal to the maximum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1.
2. The following applies for all x, y = 0, … nTbS-1:
   * If predSamples[ x ][ y ] < predAvg, a variable segIdx[ x ][ y ] is set equal to 0;
   * Otherwise (predSamples[ x ][ y ] >= predAvg), segIdx[ x ][ y ] is set equal to 1.
3. For j from 0 to nSegNum[ xTb0 ][ yTb0 ]-1, the following applies:
   * A variable segMin[ j ] is set equal to the minimum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1 satisfying segIdx[ x ][ y ] is equal to j. A variable segMax[ j ] is set equal to the maximum value for all predSamples[ x ][ y ] with x, y = 0, … nTbS-1 satisfying segIdx[ x ][ y ] is equal to j.
   * A variable segPred[ j ] is set equal to (segMin[ j ] + segMax[ j ]+ 1)>>1.
   * A variable segValue[ j ] is set equal to Clip3( 0, ( 1  << BitDepthY ) − 1 ), .SegOffset[xTb0 ][ yTb0  ][ j ][ cIdx ]+ segPred[ j ]).
4. For all x, y = 0, … nTbS-1, predSamples[ x ][ y ] is set equal to segValue[segIdx[ x ][ y ]].

…

**Parsing process**

Table 9‑4 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **ctxTable** | **initType** | | |
| **0** | **1** | **2** |
| sao( ) | sao\_merge\_left\_flag sao\_merge\_up\_flag |  | 0 | 1 | 2 |
| sao\_type\_idx\_luma sao\_type\_idx\_chroma |  | 0 | 1 | 2 |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] |  | 0..2 | 3..5 | 6..8 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag |  | 0 | 1 | 2 |
| cu\_skip\_flag |  |  | 0..2 | 3..5 |
| ~~intra\_bc\_flag[ ][ ]~~ |  | ~~0~~ | ~~1~~ | ~~2~~ |
| palette\_mode\_flag[ ][ ] |  | 0 | 1 | 2 |
| pred\_mode\_flag |  |  | 0 | 1 |
| part\_mode |  | 0 | 1..4 | 5..8 |
| prev\_intra\_luma\_pred\_flag[ ][ ] |  | 0 | 1 | 2 |
| intra\_chroma\_pred\_mode[ ][ ] |  | 0 | 1 | 2 |
| rqt\_root\_cbf |  |  | 0 | 1 |
| cu\_residual\_act\_flag |  | 0 | 1 | 2 |
| prediction\_unit( ) | merge\_flag[ ][ ] |  |  | 0 | 1 |
| merge\_idx[ ][ ] |  |  | 0 | 1 |
| intra\_bc\_flag[ ][ ] |  | 0 | 1 | 2 |
| flipping\_ibc\_flag[ ][ ] | Table 9-38’ | 0…2 | 0…2 | 0…2 |
| flipping\_ibc\_direction[ ][ ] | Table 9-38’ | 0…2 | 0…2 | 0…2 |
| segmental\_ibc\_flag [ ][ ] | Table 9-39’ | 0…2 | 0…2 | 0…2 |
| abs\_segment\_offset[ ][ ][ ][ ] | Table 9-40’ | 0…4 | 0…4 | 0…4 |
| segment\_offset\_sign[ ][ ][ ][ ] | Table 9-40’ | 0…4 | 0…4 | 0…4 |
| inter\_pred\_idc[ ][ ] |  |  | 0..4 | 5..9 |
| ref\_idx\_l0[ ][ ], ref\_idx\_l1[ ][ ] |  |  | 0..1 | 2..3 |
| mvp\_l0\_flag[ ][ ], mvp\_l1\_flag[ ][ ], bvp\_flag[ ][ ] |  |  | 0 | 1 |

| Table 9‑38 – Syntax elements and associated binarizations | | | |
| --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **Binarization** | |
| **Process** | **Input parameters** |
| coding\_quadtree( ) | split\_cu\_flag[ ][ ] | FL | cMax = 1 |
| coding\_unit( ) | cu\_transquant\_bypass\_flag | FL | cMax = 1 |
| cu\_skip\_flag | FL | cMax = 1 |
| ~~intra\_bc\_flag~~ | ~~FL~~ | ~~cMax = 1~~ |
| palette\_mode\_flag | FL | cMax = 1 |
| …… |  |  |
| palette\_coding( ) | previous\_palette\_entry\_flag[] | FL | cMax = 1 |
| …… |  |  |
| prediction\_unit( ) | merge\_flag[ ][ ] | FL | cMax = 1 |
| merge\_idx[ ][ ] | TR | cMax = MaxNumMergeCand − 1, cRiceParam = 0 |
| intra\_bc\_flag | FL | cMax = 1 |
| flipping\_ibc\_flag[ ][ ] | FL | cMax = 1 |
| flipping\_ibc\_direction[ ][ ] | FL | cMax = 1 |
| segmental\_ibc\_flag | FL | cMax = 1 |
| abs\_segment\_offset | EG0 | - |
| segment\_offset\_sign | FL | cMax = 1 |
| inter\_pred\_idc[ x0 ][ y0 ] |  | nPbW, nPbH |
| ref\_idx\_l0[ ][ ] | TR | cMax = num\_ref\_idx\_l0\_active\_minus1, cRiceParam = 0 |
| mvp\_l0\_flag[ ][ ] | FL | cMax = 1 |
| ref\_idx\_l1[ ][ ] | TR | cMax = num\_ref\_idx\_l1\_active\_minus1, cRiceParam = 0 |
| mvp\_l1\_flag[ ][ ] | FL | cMax = 1 |
| bvp\_flag[ ][ ] | FL | cMax = 1 |

Table 9‑38’ – Values of initValue for ctxIdx of flipping\_ibc\_flag and flipping\_ibc\_direction

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of sao\_merge\_left\_flag and sao\_merge\_up\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 146 | 154 | 157 |

Table 9‑39’ – Values of initValue for ctxIdx of segmental\_ibc\_flag

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of segmental\_ibc\_flag** | | |
| **0** | **1** | **2** |
| **initValue** | 154 | 154 | 154 |

Table 9‑40’ – Values of initValue for ctxIdx of abs\_segment\_offset and segment\_offset\_sign

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of** abs\_segment\_offset | | | | |
| **0** | **1** | **2** | **3** | **4** |
| **initValue** | 154 | 154 | 154 | 154 | 154 |

| Table ‑43 – Assignment of ctxInc to syntax elements with context coded bins | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax element** | **binIdx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| … |  |  |  |  |  |  |
| intra\_bc\_flag | 0 | na | na | na | na | na |
| flipping\_ibc\_flag | 0 | na | na | na | na | na |
| flipping\_ibc\_direction | 0 | na | na | na | na | na |
| segmental\_ibc\_flag[ ][ ] | 0 | na | na | na | na | na |
| abs\_segment\_offset[ x0 ][ y0 ][ j ][cIdx ] | 2\*(nSegNum[ x0 ][ y0 ]– 2) + j | bypass | bypass | bypass | bypass | bypass |
| segment\_offset\_sign[ x0 ][ y0 ][ j ][cIdx ] | 2\*(nSegNum[ x0 ][ y0 ]– 2 ) + j | na | na | na | na | na |
| pred\_mode\_flag | 0 | na | na | na | na | na |
| … |  |  |  |  |  |  |

Table 9‑44 – Specification of ctxInc using left and above syntax elements

|  |  |  |  |
| --- | --- | --- | --- |
| **Syntax element** | **condL** | **condA** | **ctxInc** |
| split\_cu\_flag[ x0 ][ y0 ] | CtDepth[ xNbL ][ yNbL ] > cqtDepth | CtDepth[ xNbA ][ yNbA ] > cqtDepth | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| cu\_skip\_flag[ x0 ][ y0 ] | cu\_skip\_flag[ xNbL ][ yNbL ] | cu\_skip\_flag[ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| flipping\_ibc\_flag[ x0 ][ y0 ] | flipping\_ibc\_flag[ xNbL ][ yNbL ] | flipping\_ibc\_flag [ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| flipping\_ibc\_direction[ x0 ][ y0 ] | flipping\_ibc\_direction[ xNbL ][ yNbL ] | flipping\_ibc\_direction [ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |
| segmental\_ibc\_flag[ x0 ][ y0 ] | segmental\_ibc\_flag[ xNbL ][ yNbL ] | segmental\_ibc\_flag[ xNbA ][ yNbA ] | ( condL  &&  availableL ) + ( condA  &&  availableA ) |