Branch Instructions

This set of slides covers the “syntactic sugar” or extended mnemonics
used to write the standard conditional branch instructions.

There are two basic branch instructions in the IBM instruction set.

```
BC    MASK,TARGET      A TYPE RX INSTRUCTON
BCR   MASK,REGISTER    A TYPE RR INSTRUCTION
```

In the Type RX instruction, the target address is computed using the base register
and displacement method, with an optional index register: **`D2(X2,B2)`**.

In the Type RR instruction, the target address is found as the contents of the register.

Each of these forms uses a four–bit mask to determine the conditions
under which the branch will be taken.

The 4–bit mask should be considered as having bits numbered left to right as 0, 1, 2, 3.

Bit 0 is the equal/zero bit.

Bit 1 is the low/minus bit.

Bit 2 is the high/plus bit.

Bit 3 is the overflow bit.

The Standard Combinations

The following table, taken from Abel's textbook, shows the standard conditional branch instructions and their translation to the BC (Branch on Condition).

The same table applies to BCR (Branch on Condition, Register).

Bit Mask Flags				Condition			
0	1	2	3				
0	0	0	0	No branch	BC 0,XX	NOP	
0	0	0	1	Bit 3: Overflow	BC 1,XX	BO XX	
0	0	1	0	Bit 2: High/Plus	BC 2,XX	BH XX	BP
0	1	0	0	Bit 1: Low/Minus	BC 4,XX	BL XX	BM
0	1	1	1	1, 2, 3: Not Equal	BC 7,XX	BNE XX	BNZ
1	0	0	0	Bit 1: Equal/Zero	BC 8,XX	BE XX	BZ
1	0	1	1	0, 2, 3: Not Low	BC 11.XX	BNL XX	BNM
1	1	0	1	0, 1, 3: Not high	BC 13,XX	BNH XX	BNP
1	1	1	1	0, 1, 2, 3: Any	BC 15,XX	B XX	

Note the two sets of extended mnemonics: one for comparisons and an equivalent set for the results of arithmetic operations.

These equivalent sets are provided to allow the assembler code to read more naturally.

The Idea of a Sort Order

Two data items of a specific data type are said to be “comparable” if they can be subjected to some sort of comparison operator with well defined results.

One common operator that can be applied to many operations is that of equality, denoted “=”. The negation of equality is inequality, denoted “≠”.

We also are interested in other comparisons, implied by what is called a “sort order”.

Given two data items of the same type, it is convenient to define three operators.

A > B if A follows B in the sort order.

A = B if A and B occupy the same place in the sort order.

A < B if A precedes B in the sort order.

Remember that each of these operators has an “opposite”.

If A > B	then not A ≤ B.	Assembler pair:	BH and BNH
If A = B	then not A ≠ B.	Assembler pair:	BE and BNE
If A < B	then not A ≥ B.	Assembler pair:	BL and BNL

Overflow: “Busting the Arithmetic”

Consider the half–word integer arithmetic in the IBM System/360. Integers in this format are 16–bit two’s complement integers with a range of

– 32,768 to 32,767

Consider the following addition problem: 24576 + 24576.
Now + 24,576 (binary 0110 0000 0000 0000) is well within the range.

0110 0000 0000 0000	24576
0110 0000 0000 0000	24576
1100 0000 0000 0000	– 16384

What happened?

We had a carry into the sign bit. This is “overflow”. The binary representation being used cannot handle the result.

NOTE: This works as unsigned arithmetic.
24,576 + 24,576 = 49,152 = 32768 + 16384.

On the System/360, such an invalid operation will set the overflow bit.

Setting the Condition Codes

We now investigate those instructions that set the condition codes.

These condition codes are set according to the sort order associated with the data type. The two main ordering schemes are numeric and EBCDIC code.

There are a number of such instructions. Here are a few.

1. **Arithmetic instructions**

   Packed decimal: ZAP, AP, SP, etc.

   Binary arithmetic: A, S, AH, SH, etc.

2. **Shift instructions**

   Packed decimal: SRP (Does not set the overflow bit)

   Register shift: SRA, SLA, SRDA, SLDA, etc.

3. **Comparison instructions. None of these can set the overflow bit.**

   Character comparison: CLC uses the EBCDIC code. “a” < “z” < “A” < “Z”.

   Packed decimal comparison: CP

   Binary comparison: C, CH, CR