Instruction Formats

There are six instruction formats, five of which are of use to us.
The sixth format is used only by code run by the Operating system.

These formats are classified by length in bytes, use of the base registers,
and object code format.

Format Name	Length in bytes	Use
RR	2	Register to register transfers.
RS	4	Register to storage and register from storage
RX	4	Register to indexed storage and register from indexed storage
SI	4	Storage immediate
SS	6	Storage–to–Storage. These have two variants, each of which we shall discuss soon.

The student will note that the main motivation for this proliferation of instruction formats seems to have been the desire to minimize the total size of the code.

Remember that this code might have to execute in 32 kilobytes of memory.

The Object Code Format

Here is a table summarizing the formats of the five instruction types.

Format	Length	Explicit operand form						
			1	2	3	4	5	6
RR	2	R1,R2	OP	R_1 R_2				
RS	4	R1,R3,D2(B2)	OP	R_1 R_3	B_2 D_2	D_2D_2		
RX	4	R1,D2(X2,B2)	OP	R_1 X_2	B_2 D_2	D_2D_2		
SI	4	D1(B1),I2	OP	I2	B_1 D_1	D_1D_1		
SS(1)	6	D1(L,B1),D2(B2)	OP	L	B_1 D_1	D_1D_1	B_2 D_2	D_2D_2
SS(2)	6	D1(L1,B1),D2(L2,B2)	OP	L_1 L_2	B_1 D_1	D_1D_1	B_2 D_2	D_2D_2

NOTES: OP is the 8–bit operation code.

R_1 R_2 and R_1 X_2 each denote two 4–bit fields to specify two registers.

The two byte entry of the form B D D D denotes a 4–bit field to specify a base register and three 4–bit fields (12 bits) to denote a 12–bit displacement.

L denotes an 8–bit field for operand length (256 bytes maximum).
L_1 and L_2 each denote a 4–bit field for an operand length (16 bytes max.).

I denotes an 8–bit (one byte) immediate operand. These are useful.

RR (Register–to–Register) Format

This is a two–byte instruction of the form **`OP R1,R2`**.

Type	Bytes			
RR	2	R1,R2	OP	R_1 R_2

The first byte contains the 8–bit instruction code.

The second byte contains two 4–bit fields, each of which encodes a register number.

This instruction format is used to process data between registers.

Here are some examples.

`AR 6,8`	**`1A 68`**	Adds the contents of register 8 to register 6.
`AR 10,11`	**`1A AB`**	Adds the contents of register 11 to register 10.
`AR R6,R8`	**`1A 68`**	Due to the standard Equate statements we use in our program assignments, **`R6`** stands for **`6`** and **`R8`** for **`8`**.

RR (Register–to–Register) Format: Branch Instructions

There are two formats used with conditional branching instructions.

The BCR (Branch on Condition Register) instruction uses a modified form of the RR format. The BC (Branch on Condition) uses the RX format.

The BCR instruction is a two–byte instruction of the form **`OP M1,R2`**.

Type	Bytes			
RR	2	M1,R2	07	M_1 R_2

The first byte contains the 8–bit instruction code, which is **`X'07'`**.

The second byte contains two 4–bit fields.

The first 4–bit field encodes a branch condition

The second 4–bit fields encodes the number of the register containing the target address for the branch instruction.

For example, the instruction **`BR R8`** is the same as **`BCR 15,R8`**.

The object code is **`07 F8.`** Branch unconditionally to the address in register 8.

We shall discuss the **`BC`** and **`BCR`** instructions in more detail at a later lecture.

RS (Register–Storage) Format

This is a four–byte instruction of the form **`OP R1,R3,D2(B2)`**.

Type	Bytes		1	2	3	4
RS	4	R1,R3,D2(B2)	OP	R_1 R_3	B_2 D_2	D_2D_2

The first byte contains the 8–bit instruction code.

The second byte contains two 4–bit fields, each of which encodes a register number.

Note that some RS format instructions use only one register, here R3 is set to 0.
In this instruction format, “0” is taken as no register, rather than register R0.

The third and fourth byte contain a 4–bit register number and 12–bit displacement, used to specify the memory address for the operand in storage.

Recall that each label in the assembly language program references an address, which must be expressed in the form of a base register with displacement.

Any address in the format of base register and displacement will appear in the form.

B D_1	D_2 D_3

B is the hexadecimal digit representing the base register.

The three hexadecimal digits D_1 D_2 D_3 form the 12–bit displacement.

RS (Register–Storage) Format Example

This is a four–byte instruction of the form **`OP R1,R3,D2(B2)`**.

Type	Bytes		1	2	3	4
RS	4	R1,R3,D2(B2)	OP	R_1 R_3	B_2 D_2	D_2D_2

1. Load Multiple Operation code = **`X'98'`**.

   Suppose the label FW3 (supposedly holding three 32–bit full–words) is at an address specified by offset **`X'100'`** from base register **`R7`**. Then we have

   `LM R5,R7,FW3    98 57 71 00`

Unpacking the object code, we again find the parts.

The operation code is **`X'98'`**, which indicates a multiple register load.

The next byte has value **`X'57'`**, which indicates two registers: **`R5`** and **`R7`**. Here it is used to represent a range of three registers: **`R5`**, **`R6`**, and **`R7`**.

The last two bytes contain the address of the label **`FW3`**. The two bytes **`71 00`** indicate

1) that the base address is contained in register **`R7`**, and
2) that the displacement from the base address is **`X'100'`**.

Another RS (Register–Storage) Format Example

This is a four–byte instruction of the form **`OP R1,R3,D2(B2)`**.

Type	Bytes		1	2	3	4
RS	4	R1,R3,D2(B2)	OP	R_1 R_3	B_2 D_2	D_2D_2

2. Shift Left Logical Operation code = **`X‘89’`**

This is also a type RS instruction, though the appearance of a typical use seems to deny this. Consider the following instruction which shifts R6 left by 12 bits.

```
SLL R6, 12    Again, I assume we have set R6 EQU 6
```

The deceptive part concerns the value 12, used for the shift count. Where is that stored?

The answer is that it is not stored, but assembled in the form of a displacement of 12 to a base register of 0, indicating that no base register is used.

The above would be assembled as **`89 60 00 0C  Decimal 12 is X‘C’`**

Here are three lines from a working program I wrote on 2/23/2009.

```
000014 5840 C302     00308    47          L     R4,=F'1'
000018 8940 0001     00001    48          SLL   R4,1
00001C 8940 0002     00002    49          SLL   R4,2
```

RX (Register–Indexed Storage) Format

This is a four–byte instruction of the form **`OP R1,D2(X2,B2)`**.

Type	Bytes		1	2	3	4
RX	4	R1,D2(X2,B2)	OP	R_1 X_2	B_2 D_2	D_2D_2

The first byte contains the 8–bit instruction code.

The second byte contains two 4–bit fields, each of which encodes a register number.

In order to illustrate this, consider the following data layout.

```
FW1  DC F'31'

     DC F'100'   Note that this full word is not labeled
```

Suppose that FW1 is at an address defined as offset **`X'123'`** from register 12.
As hexadecimal **`C`** is equal to decimal 12, the address would be specified as **`C1 23`**.

The next full word might have an address specified as **`C1 27`**, but we shall show another way to do the same thing. The code we shall consider is

```
L  R4,FW1     Load register 4 from the full word at FW1

AL R4,FW1+4   Add the value at the next full word address
```

RX (Register–Indexed Storage) Format (Continued)

This is a four–byte instruction of the form **`OP R1,D2(X2,B2)`**.

Type	Bytes		1	2	3	4
RX	4	R1,D2(X2,B2)	OP	R_1 X_2	B_2 D_2	D_2D_2

Consider the two line sequence of instructions

```
L  R4,FW1      Operation code is X'58'.

AL R4,FW1+4    Operation code is X'5E'.
```

The load instruction, remembering that the address of FW1 is specified as **`C1 23`**. The base register is R12, the displacement is **`X'123'`**, and there is no index register; so we have

```
58 40 C1 23
```

The next instruction is similar, except for its operation code.

```
5E 40 C1 27
```

NOTE: In each of the examples above, the 4–bit value X2 = 0. When a 0 is found in the index position, that indicates that indexed addressing is not used. Register 0 cannot be used as either a base register or an index register.

RX Format (Using an Index Register)

Here we shall suppose that we want register 7 to be an index register.

As the second argument is at offset 4 from the first, we set R7 to have value 4.

This is a four–byte instruction of the form **`OP R1,D2(X2,B2)`**.

Type	Bytes		1	2	3	4
RX	4	R1,D2(X2,B2)	OP	R_1 X_2	B_2 D_2	D_2D_2

Consider the three line sequence of instructions

```
L  R7,=F‘4’   Register 7 gets the value 4.
L  R4,FW1     Operation code is X‘58’.
AL R4,FW1(R7)   Operation code is X‘5E’.
```

The object code for the last two instructions is now.

```
58 40 C1 23  This address is at displacement 123
             from the base address, which is in R12.
             Note X2 = 0, indicating no indexing.
5E 47 C1 23  R7 contains the value 4.
             The address is at displacement 123 + 4
             or 127 from the base address, in R12.
```

More on “Index Register 0”

Consider the instruction

```
L  R4,FW1     Operation code is X‘58’.
```

The object code for this instruction is of the form

```
58 40 C1 23
```

The second byte of the instruction has the destination register set as 4 (either decimal or hexadecimal, you choose), and the “index register” set to 0.

The intent of the instruction is that indexed addressing not be invoked.

There are two common ways to handle this addressing procedure.

1. The solution chosen by IBM is that the 0 indicates “do not index”, and that the value of register R0 is not used or changed.

2. Another common solution, tried as early as the CDC–6600, is to specify that register R0 stores the constant 0; R0 ≡ 0.

   The 0 in the index register position would then indicate “index by R0”, that is to add 0 to the base–displacement address; in other words, no indexing.

Each method has its advantages. The second simplifies design of the control unit.

RX Format (Branch on Condition)

The **BC** (Branch on Condition) is a 4–byte instruction of the form **OP M1,D2(X2,B2)**. Its operation code is **X'47'**.

Type	Bytes		1	2	3	4
RX	4	R1,D2(X2,B2)	**47**	M_1 X_2	B_2 D_2	D_2D_2

The first byte contains the 8–bit instruction code, which is **X'47'**.

The second byte contains two 4–bit fields.

The first four bits contain the mask for the branch condition codes

The second four bits contain the number of the index register used in computing the address of the jump target.

The next two bytes contain the 4–bit number of the base register and the 12–bit displacement used to form the unindexed address of the branch target.

Suppose that address **TARGET** is formed by offset **X'666'** using base register 8. No index is used and the instruction is **BNE TARGET**, equivalent to **BC 7,TARGET**, as the condition mask for "Not Equal" is the 4–bit number **0111**, or decimal 7.

The object code for this is **47 70 86 66**.

SI (Storage Immediate) Format

This is a four–byte instruction of the form **`OP D1(B1),I2`**.

Type	Bytes		1	2	3	4
SI	4	D1(B1), I2	OP	I2	B_1 D_1	D_1D_1

The first byte contains the 8–bit instruction code.

The second byte contains the 8–bit value of the second operand, which is treated as an **immediate operand**. The instruction contains the **value** of the operand, not its address.

The first operand is an address, specified in standard base register and displacement form.

Two instances of the instruction are :

`MVI` Move Immediate

`CLI` Compare Immediate

Suppose that the label **`ASTER`** is associated with an address that is specified using register **`R3`** as a base register, with **`X'6C4'`** as offset.

The operation code for **`MVI`** is **`X'92'`** and the EBCDIC for "**`*`**" is **`X'5C'`**.

`MVI ASTER,C '*'` is assembled as **`92 5C 36 64`**.

The Storage–to–Storage Instructions

There are two formats for the SS (Storage–to–Storage) instructions.

Each of the formats requires six bytes for the instruction object code.

The two types of the SS instruction are as follows:

1. **The Character Instructions**

   These are of the form **`OP D1(L,B1),D2(B2)`**, which provide a length for only operand 1. The length is specified as an 8–bit byte.

   Examples: **`MVC`** Move Characters
   `CLC` Compare Characters

2. **The Packed Decimal Instructions**

   These are of the form **`OP D1(L1,B1),D2(L2,B2)`**, which provide a length for each of the two operands. Each length is specified as a 4–bit hexadecimal digit.

   Examples: **`ZAP`** Zero and Add Packed (Move Packed)
   `AP` Add Packed
   `CP` Compare Packed

Storage–to–Storage: Length Fields

Consider the two formats used to store a length in bytes.

These are a four–bit hexadecimal digit and an eight–bit byte.

Four bits will store an unsigned integer in the range 0 through 15.

Eight bits will store an unsigned integer in the range 0 through 255.

However, a length of 0 bytes is not reasonable for an operand.

For this reason, the value stored is the one less than the length of the operand.

Field Size	Value Stored	Operand Length	
Four bits	0 – 15	1 – 16	bytes
Eight bits	0 – 255	1 – 256	bytes

By examination of all instruction formats, we can show that only the SS (Storage–to–Storage) format instructions require length codes.

Storage–to–Storage: Character Instructions

These are of the form **`OP D1(L,B1),D2(B2)`**, which provide a length for only operand 1. The length is specified as an 8–bit byte.

Type	Bytes	Form	1	2	3	4	5	6
SS(1)	6	D1(L,B1),D2(B2)	OP	L	B_1 D_1	D_1D_1	B_2 D_2	D_2D_2

The first byte contains the operation code, say **`X'D2'`** for **`MVC`** or **`X'D5'`** for **`CLC`**.

The second byte contains a value storing one less than the length of the first operand, which is the destination for any move.

Bytes 3 and 4 specify the address of the first operand, using the standard base register and displacement format.

Bytes 5 and 6 specify the address of the second operand, using the standard base register and displacement format.

It is quite common for both operands to use the same base register.

Example of Character Instructions

The form is **`OP D1(L,B1),D2(B2)`**. The object code format is as follows:

Type	Bytes	Form	1	2	3	4	5	6
SS(1)	6	D1(L,B1),D2(B2)	OP	L	B_1 D_1	D_1D_1	B_2 D_2	D_2D_2

Consider the example assembly language statement, which moves the string of characters at label **`CONAME`** to the location associated with the label **`TITLE`**.

```
MVC TITLE,CONAME
```

Suppose that:

1. There are fourteen bytes associated with **`TITLE`**, say that it was declared as **`TITLE DS CL14`**. Decimal 14 is hexadecimal E.
2. The label **`TITLE`** is referenced by displacement **`X‘40A’`** from the value stored in register **`R3`**, used as a base register.
3. The label **`CONAME`** is referenced by displacement **`X‘42C’`** from the value stored in register **`R3`**, used as a base register.

Given that the operation code for MVC is **`X‘D2’`**, the instruction assembles as

```
D2 0D 34 0A 34 2C   Length is 14 or X‘0E’; L – 1 is X‘0D’
```

Storage–to–Storage: Packed Decimal Instructions

These are of the form **`OP D1(L1,B1),D2(L2,B2)`**, which provide a 4–bit number representing the length for each of the two operands.

Type	Bytes	Form	1	2	3	4	5	6
SS(2)	6	D1(L1,B1),D2(L2,B2)	OP	L_1 L_2	B_1 D_1	D_1D_1	B_2 D_2	D_2D_2

The first byte contains the operation code, say **`X'FA'`** for **`AP`** or **`X'F9'`** for **`CP`**.

The second byte contains a two values, each a 4–bit binary number (one hex digit).

L1 A value that is one less than the length of the first operand.

L2 A value that is one less than the length of the second operand.

Bytes 3 and 4 specify the address of the first operand, using the standard base register and displacement format.

Bytes 5 and 6 specify the address of the second operand, using the standard base register and displacement format.

IBM will frequently call these **decimal instructions**. Here are two lines from the standard reference card, officially called FORM GX20–1850.

AP Decimal Add CP Compare Decimal

Example of Packed Decimal Instructions

The form is **`OP D1(L1,B1),D2(L2,B2)`**. The object code format is as follows:

Type	Bytes	Form	1	2	3	4	5	6
SS(2)	6	D1(L1,B1),D2(L2,B2)	OP	L_1 L_2	B_1 D_1	D_1D_1	B_2 D_2	D_2D_2

Consider the assembly language statement below, which adds **`AMOUNT`** to **`TOTAL`**.

```
AP TOTAL,AMOUNT
```

Assume:

1. **`TOTAL`** is 4 bytes long, so it can hold at most 7 digits.
2. **`AMOUNT`** is 3 bytes long, so it can hold at most 5 digits.
3. The label **`TOTAL`** is at an address specified by a displacement of **`X'50A'`** from the value in register **`R3`**, used as a base register.
4. The label **`AMOUNT`** is at an address specified by a displacement of **`X'52C'`** from the value in register **`R3`**, used as a base register.

The object code looks like this: **`FA 32 35 0A 35 2C`**

Example of Packed Decimal Instructions (Continued)

The form is **`OP D1(L1,B1),D2(L2,B2)`**. The object code format is as follows:

Type	Bytes	Form	1	2	3	4	5	6
SS(2)	6	D1(L1,B1),D2(L2,B2)	OP	L_1 L_2	B_1 D_1	D_1D_1	B_2 D_2	D_2D_2

Consider **`FA 32 35 0A 35 2C.`** The operation code **`X'FA'`** is that for the Add Packed (Add Decimal) instruction, which is a type SS(2). The above format applies.

The field **`32`** is of the form L_1 L_2.

The first value is **`X'3'`**, or 3 decimal. The first operand is 4 bytes long.

The second value is **`X'2'`**, or 2 decimal. The second operand is 3 bytes long.

The two–byte field **`35 0A`** indicates that register 3 is used as the base register for the first operand, which is at displacement **`X'50A'`**.

The two–byte field **`35 2C`** indicates that register 3 is used as the base register for the second operand, which is at displacement **`X'52C'`**.

It is quite common for both operands to use the same base register.