VisibleZ Lesson 1 – Starting to Write Object Code

A machine is organized like an onion with layers of software covering the hardware below. To an application programmer, it appears that the lowest level of hardware provides a machine language of 0's and 1's with typically hundreds of available instructions (on System/z there are of over 1200 instructions). In truth, the machine instructions are also programmed at an even lower level that we will ignore. At the machine language level (object code), we can build programs written as long strings of binary 1's and 0's. As you can imagine, this is rather tedious. To provide some relief we could organize binary strings into groups of four digits and represent each group as a hexadecimal digit:

Binary String:	00001010101111010101111100010111
Organize into groups of 4:	0000|1010|1011|1101|0101|1111|0001|0111
Replace each group with a hex digit:	0 A B D 5 F 1 7

Since a byte is eight bits, a byte can be represented a two hex digits: 11100001 = E1 in hexadecimal.

VisibleZ can read and interpret object code programs. The format for VisibleZ programs is two hexadecimal digits per byte with each byte separated by a space for readability. Everything is in a character format and is case insensitive. VisibleZ displays hex characters in lower case, so the binary string above would look like this in memory: **0a bd 5f 17**. You can create object code programs with any text editor.

Where Am I?

One of the first things a program has to do when it starts executing is to find out where in memory it is located. The System/z instruction for doing this is called **Branch and Save Register**. In assembly language (one step up from object code) it is called **BASR**, and has two operands which are registers, so it might be coded like this: BASR 12,0 in assembly language. A register is a machine component that can hold 64-bits and can perform high-speed binary arithmetic. Registers are often used to hold addresses, and we use them to help us locate data that is stored in the memory of the machine. In memory, the BASR 12,0 instruction would occupy two bytes. The first byte would identify the instruction itself, and is called the operation code, or opcode for short. The opcode for BASR is 0d in hexadecimal. There are 16 general purpose registers in System/z. Conveniently, they are numbered 0-15, so we can represent a register using a single hex digit. BASR uses two registers, so **BASR 12,0** looks like **0d c0** in VisibleZ.

When we execute **BASR 12,0**, the address of the instruction that follows BASR (the next instruction) is loaded into the first register, 12. This is how a program discovers where it is in memory. The second operand register 0 is an indication that we don't want to branch at all. If we had executed **BASR 12,8**, the machine would use the address in register 8 to find the next instruction it would execute. By choosing register 0, execution continues with the instruction following **BASR 12,0**.

Try running program **basr.obj** (found in the Codes directory) in VisibleZ. This program contains a sequence of BASR's. Each time you "cycle" the current instruction, the program loads a new address into register 12 and then moves on to the next instruction. At the end if you keep cycling, the program loops back to the start the process again.

Jumping Back

The instruction at the end of the program is called **Branch on Condition Register (BCR)**. Like Branch and Save Register, it consists of two bytes (both these instructions have the same format called **Register to Register (RR)**). The difference is that in **BCR**, the first operand is a mask that describes the conditions under which we will branch

– for the moment we will assume that coding **15=f** means we should always branch. Where are we branching? That is determined by operand 2 which is a register. Whatever address is contained in operand 2 becomes the target address for the branch. If you code **BCR 15,7**, we will branch to the address stored in register 7. The opcode for **BCR** is 07, so BCR 15,7 would look like this in VisibleZ: **07 f7**.

Your first program in VisibleZ

Write a program with these two instructions. Perform a series of BASRs that loads registers 4, 5, and 6 with an address. At the end of the program code a BCR that jumps back to the second BASR. Why can't you jump back to the first BASR, given what we know so far?

When an assembler program starts to execute, register 15 contains the address of the first instruction in your program. Use this fact to modify your program to jump back to the beginning BASR.

Looking back and learning more

These two instructions, BASR and BCR, are both Register to Register (RR) instructions. Each instruction occupies two bytes in memory:

BASR: `OPCODE | R1R2`

BCR: `OPCODE | M1R2`

M1 is a mask that determines the branch conditions (there are four). Here are some example mask settings:

```
Equal/Zero  Low/Minus High/Plus Overflow   (1 – branch, 0 – no branch)
     1          1         1         1       = x'F'  Branch on all conditions
     1          0         0         0       = x'8'  Branch if equal or zero
     0          1         0         0       = x'4'  Branch if low
```

The mask is compared to the two bit condition code. If the mask contains a 1-bit that matches the condition code, the branch occurs. Run your program again and look for the condition code when the BCR is executed.