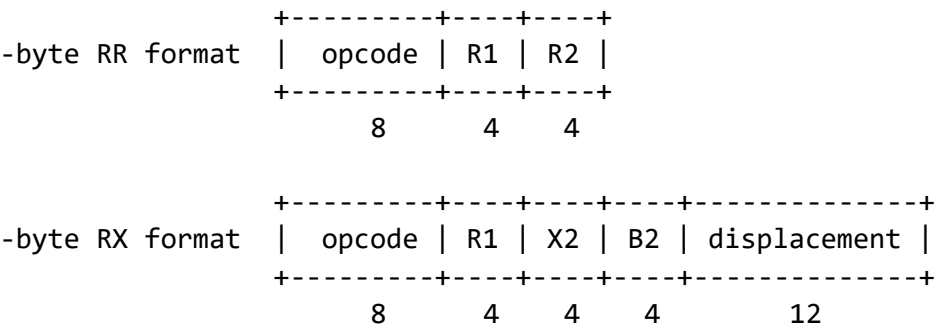
IBM S/360 subset simulation, updated draft assignment of Feb. 10, 2022

Sixteen 32-bit general registers, addressed as 0 to 0xF
24-bit instruction address
2-bit condition code, implicitly used in several instructions
  When set by add and subtract:
    0 => result is 0
    1 => result is < 0
    2 => result is > 0
    (3 indicates overflow, which we will not implement)
  When set by compare (treating both operands as 32-bit signed integers):
    0 => operands are equal
    1 => first operand is low
    2 => first operand is high
    (3 is not used)

Byte-addressable memory with 24-bits addresses
  (we will limit our simulation to a 4096-byte memory)
Memory is aligned
  An instruction starts on an address that is a multiple of 2
  A data word starts on an address that is a multiple of 4

We will implement the two basic instruction formats



Effective address calculation = content of X2 + content of B2 + displacement, unless X2 or B2 is specified as a 0, then that component is not used (note that both X2 and B2 can be specified as 0 and then the displacement by itself is the effective address); the effective address should be clamped to 24 bits

For branch on condition, R1 becomes a 4-bit mask that selects one or more condition code values in this manner:
  First mask bit selects condition code value 0
  Second mask bit selects condition code value 1
  Third mask bit selects condition code value 2
  Fourth mask bit selects condition code value 3

Thus a mask of all ones (0xF) results in an unconditional branch and a mask of zero results in a no-op (no operation).

RR opcodes
18 LR - Load Register - load the contents of R2 into R1
19 CR - Compare Register – compare the contents of R1 with the
         contents of R2 and set the condition code
1A AR - Add Register - add the contents of R2 to the contents of R1,
         place the result into R1, and set the condition code
         (i.e., R1 = R1 + R2)
1B SR - Subtract Register - subtract the contents of R2 from the
         contents of R1, place the result into R1, and set the
         condition code (i.e., R1 = R1 – R2)

RX opcodes
41 LA - Load Address - load the 24-bit effective address into R1 and
         set the high 8 bits to zeros (there is no alignment check
         nor range check on the address; it is treated as an
         immediate value that will be loaded into the register)
46 BCT - Branch on Count - decrement the contents of R1 and branch to
         the effective address if the result is not equal to zero
         (the effective address is calculated before the decrement)
47 BC - Branch on Condition - treat the R1 field as a branch mask
         and branch to the effective address if the current
         condition code value is one of the values selected by
         the mask bit(s)
50 ST - Store - store the contents of R1 into the memory word at the
         effective address (align check and range check the address)
58 L   - Load - load the contents of the memory word at the effective
         address into R1 (align check and range check the address)
59 C   - Compare – compare the contents of R1 with the contents of
         the memory word at the effective address and set the
         condition code (align check and range check the address)

A S/360 Principles of Operation manual can be found at
http://bitsavers.trailing-edge.com/pdf/ibm/360/princOps/A22-6821-0_360PrincOps.pdf

Your program should

- Implement a verbose mode of output using a –v command line parameter
- Read bytes as two hexadecimal digits at a time from stdin
- Echo bytes the bytes read from stdin in verbose mode
- Start the simulated instruction execution at address 0
- Stop when an opcode of zero is encountered (that is, treat as a halt)
- Check for unaligned and out of range instruction and data addresses

(Note that the LA instruction is allowed to load a 24-bit value)

- Print instruction information and updated register values in verbose mode
- Print final memory contents in word format in verbose mode
  (Note that you should print as many words as needed to show all the byte locations that were loaded from the input)
- Track and print dynamic execution statistics at the end of the simulation

Simple loop example (will need to be hand-assembled)

```
      LA  R5,0,0,3      load R5 with loop count
Loop: SR  R6,R6         sequence to set R6 = 4 * R5
      AR  R6,R5
      AR  R6,R6
      AR  R6,R6
      ST  R5,R6,0,0x14  store count into an array of memory words at 0x14
      BCT R5,0,0,Loop   branch on count in R5
      Halt              (not counted as an instruction)
```

Input file in ASCII text

```
41 50 00 03
1B 66
1A 65
1A 66
1A 66
50 56 00 14
46 50 00 04
00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
```

Example code related to BCT instruction

```
…
#define INST_BCT 5
…

int reg[16] = {0};
…
unsigned char memory[4096] = {0};
```

```
…

// decode extracts the opcode, r1, and r2/x2/mask fields, and for RX
//   instructions, it also extracts the b2 field and calculates the
//   effective address

…

  if( verbose ){
    printf( "\nBCT instruction, " );
    printf( "operand 1 is R%x, ", r1 );
    printf( "branch target is address %06x\n", eff_addr );
  }
  reg[r1]--;
  if( reg[r1] != 0 ){
    inst_addr = eff_addr;
    bct_taken_cnt++;
  }
  inst_cnt[INST_BCT]++;

…

  printf( "    BCT instructions  = %d", inst_cnt[INST_BCT] );
  if( inst_cnt[INST_BCT] > 0 ){
    printf( ", taken = %d (%.1f%%)\n", bct_taken_cnt,
      100.0*((float)bct_taken_cnt)/((float)inst_cnt[INST_BCT]) );
  }else{
    printf("\n");
  }
```