Name: **Evan Hammer**
**Marcus Berger**
**Kevin Hilt**

**CSC 317 Program**
**Program #2**
**Title: B-17**
**Due: May 1, 2015**

---

## 0.1 Program Description

The B17 is a 24-bit word-addressable accumulator architecture. It supports up to 64 instructions, up to four addressing modes, and 212 (4096) words of memory. Main memory consists of 4,096 words, each of which is 24 bits wide. Addresses are 12 bits wide, and range from 000 through FFF. Within the word, bits are numbered from right to left, with bit 0 the least significant bit and bit 23 the most significant bit. The memory interface transfers one word of data at a time. Communication with memory is via two registers, MAR and MDR. Other locations include IC, ABUS, DBUS three registers, AC, ALU and IR. The B17 uses fixed-length instructions. Instructions have the following format an ADDR a twelve-bit operand address (bits 23-12), a six-bit opcode (bits 11-6), and a six-bit addressing mode (bits 5-0). The b17 uses those bits to determine the instruction to be executed from the instruction set. The program continues to execute instructions untill a halt instruction is found or a halting error occurs.

## 0.2 Group Assesment

Kevin Hilt: 35%
Marcus Berger: 35%
Evan Hammer: 30%

## 0.3 Libraries used

Libraries used in this program are :
$\langle iostream \rangle, \langle iomanip \rangle, \langle string \rangle, \langle sstream \rangle, \langle fstream \rangle, and \langle cstdlib \rangle$

## 0.4 Algorithms

1. Read in the object file and set up the memory array
2. Read in instruction counter and the number of instruction to be executed
3. For each instruction begin to execute instruction
   (a) Decode the instruction and determine the address mode and instruction to be executed.
   (b) Execute the correct instruction
4. Check the address mode to get the correct output

5. Output the information for executed instruction

6. Repeat steps 3-5 for each instruction

7. If an error is encountered print correct halt message and exit program

## 0.5 Functions and Program Structure:

int main(int argc, char argv[]) - The main function handles file I/O and output formating for the b17 program. Written by Marcus Beger, Evan Hammer, Kevin Hilt using a paired programming approach

void get_address_mode(int IR, int &ABUS) - This function pulls out the bits containing the address mode out of the instruction and sets the ABUS equal to the address mode Written by Marcus Berger

void get_instruction(int IR, string &instruction) - This function pulls out the bits containing the instruction and sets it equal to a value at an index into an array containing the instruction set. Written by Marcus Beger, Evan Hammer, Kevin Hilt using a paired programming approach

void hex_to_int(string hex_string, int& stored_int) - This function converts a hex number string to an int variable. Written by Kevin Hilt

void read_memory(int memory[], int instruction_number, int MAR, int &MDR, char* filename) - This function pulls out the bits containing the address mode out of the instruction and sets the ABUS equal to the address mode. Written by Marcus Beger, Evan Hammer, Kevin Hilt using a paired programming approach

void match_instruction(int memory[], int &MAR, int &AC, int &DBUS, int &ABUS, int &IC, int &IR, ofstream &output, string instruction) -This function contains the various code to execute each instruction and the if statements to determine which one to do. Written by Marcus Beger, Evan Hammer, Kevin Hilt using a paired programming approach

### 0.5.1 Compiling and Usage:

Compile by typing make b17 in Linux
Usage: b17 object file name
Sample object file:

```
50 1 000000
c4 5 050404 200800 300800 102840 050c00
101 2 300 9
200 1 30
300 1 10
c4
```

Output file will contain memory address, instructions, the instruction executed, the address mode, that contains of the accumulator and the contains of registers x0-x3 for each instruction executed. When a halt instruction or halt error is encountered a correct message is printed and the program terminates.
Sample output file:

```
0c4: 050404 LD IMM AC[000050] X0[000] X1[000] X2[000] X3[000]
0c5: 200800 ADD 200 AC[000080] X0[000] X1[000] X2[000] X3[000]
0c6: 300800 ADD 300 AC[000090] X0[000] X1[000] X2[000] X3[000]
0c7: 102840 SUB 102 AC[000087] X0[000] X1[000] X2[000] X3[000]
0c8: 050c00 J 050 AC[000087] X0[000] X1[000] X2[000] X3[000]
050: 000000 HALT AC[000087] X0[000] X1[000] X2[000] X3[000]
Machine Halted - HALT instruction executed
```

### 0.5.2 Test

Test 1: Test ADD, SUB, Jump, Load, and Halt instructions with immediate and direct addressing modes input:

```
50 1 000000
c4 5 050404 200800 300800 102840 050c00
101 2 300 9
200 1 30
300 1 10
c4
```

Output:

```
0c4: 050404 LD IMM AC[000050] X0[000] X1[000] X2[000] X3[000]
0c5: 200800 ADD 200 AC[000080] X0[000] X1[000] X2[000] X3[000]
0c6: 300800 ADD 300 AC[000090] X0[000] X1[000] X2[000] X3[000]
0c7: 102840 SUB 102 AC[000087] X0[000] X1[000] X2[000] X3[000]
0c8: 050c00 J 050 AC[000087] X0[000] X1[000] X2[000] X3[000]
050: 000000 HALT AC[000087] X0[000] X1[000] X2[000] X3[000]
Machine Halted - HALT instruction executed
```

Text 2: Test illegal addressing mode input:

```
50 1 000000
100 5 200400 201840 202800 005844 050c84
200 3 30 31 10
100
```

Output:

```
100: 200400 LD   200  AC[000030] X0[000] X1[000] X2[000] X3[000]
101: 201840 SUB  201  AC[ffffff] X0[000] X1[000] X2[000] X3[000]
102: 202800 ADD  202  AC[00000f] X0[000] X1[000] X2[000] X3[000]
103: 005844 SUB  IMM  AC[00000a] X0[000] X1[000] X2[000] X3[000]
104: 050c84 JN   ???  AC[00000a] X0[000] X1[000] X2[000] X3[000]
Machine Halted - illegal addressing mode.
```

Test 3: Test clear instruction and unimplemented op code Input:

```
50 1 000000
ff 7 075400 040804 077440 005884 076400 077800 030a04
75 3 30 20 10
ff
```

Output:

```
0ff: 075400 LD   75   AC[000030] X0[000] X1[000] X2[000] X3[000]
100: 040804 ADD  IMM  AC[000070] X0[000] X1[000] X2[000] X3[000]
101: 077440 ST   77   AC[000070] X0[000] X1[000] X2[000] X3[000]
102: 005884 CLR  IMM  AC[000000] X0[000] X1[000] X2[000] X3[000]
103: 076400 LD   76   AC[000020] X0[000] X1[000] X2[000] X3[000]
104: 077800 ADD  77   AC[000090] X0[000] X1[000] X2[000] X3[000]
105: 030a04 ADDX IMM  AC[000090] X0[000] X1[000] X2[000] X3[000]
Machine Halted - unimplemented opcode
```

### 0.6 What was submitted?

b17.cpp contains the main function for the b17 program
b17_functions.cpp contains the get_instruction, get_addressmode, hex_to_int, read_memoy, and match_instruction functions to prevent needing to repeat code,
b17.h header file containing libraries used and function prototypes for the b17 program
makefile complies the b17 program by "typing make b17"
HammerBergerHiltB17.pdf