```c
/*
  alu−main.c
  bho1 2006; bho1 19.11.2006; bho1 8.12.2007
  bho1 29.11.2007 : init value for rega, regb, accu. flags to full 16 Bit
  bho1 19.11.2009 : corrected error with neg_b calling neg_a
  bho1 10.12.2009 : corrected error where neg_b and not_b loaded arg into rega
  bho1 5.7.2011
  renamed alu_exec_line to alu_parse_line
  bho1 10.10.2011
  functional appoach: alu(ALU_OP_CODE, rega, regb, accumulator, flags);
  GPL
*/
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "register.h"
#include "alu−opcodes.h"
#include "alu.h"

// the registers
char rega[REG_WIDTH+1] = "01234567";
char regb[REG_WIDTH+1] = "01234567";
char accumulator[REG_WIDTH+1] = "01234567";
char flags[REG_WIDTH+1] = "01234567";

/*
  Reset ALU
  resets registers and calls alu_op_reset
*/
void alu_main_reset(char rega[], char regb[], char accumulator[], char flags[]){
  int i;
  /* clear rega, regb, accumulator, flags */
  for(i=0; i<REG_WIDTH; i++){
    rega[i] = ’0’;
    regb[i] = ’0’;
    accumulator[i] = ’0’;
    flags[i] = ’0’;
  }
}

/*
  take string cmd_line, parse the line and call the alu function
  corresponding to alu−opcodes.h
*/
void alu_parse_line(char *cmd_line){
  char opcode[100];
  char operand1[100];
  char operand2[100];
  int nargs = 0;

  nargs = sscanf (cmd_line, "%s %s %s", opcode, operand1, operand2);

  switch(nargs){
  case 3:
        ldhex2register(operand1, rega);
        ldhex2register(operand2, regb);
        if(!strcmp(opcode,"add")){
          alu(ALU_OP_ADD, rega, regb, accumulator, flags);
        }
        if(!strcmp(opcode,"sub"))
          alu(ALU_OP_SUB, rega, regb, accumulator, flags);
        if(!strcmp(opcode,"and"))
          alu(ALU_OP_AND, rega, regb, accumulator, flags);
```

```c
        if(!strcmp(opcode,"or"))
          alu(ALU_OP_OR, rega, regb, accumulator, flags);
        if(!strcmp(opcode,"xor"))
          alu(ALU_OP_XOR, rega, regb, accumulator, flags);
        printf("%s %s %s\n", opcode, operand1, operand2);
        break;
  case 2:
        ldhex2register(operand1, rega);
        if(!strcmp(opcode,"neg_a"))
      alu(ALU_OP_NEG_A, rega, regb, accumulator, flags);
    if(!strcmp(opcode,"neg_b"))
      alu(ALU_OP_NEG_B, rega, regb, accumulator, flags);
    if(!strcmp(opcode,"not_a"))
          alu(ALU_OP_NOT_A, rega, regb, accumulator, flags);
    if(!strcmp(opcode,"not_b"))
      alu(ALU_OP_NOT_B, rega, regb, accumulator, flags);
        printf("%s %s\n", opcode, operand1);
        break;
  case 1:
    if(!strcmp(opcode,"reset")){
          alu_main_reset(rega, regb, accumulator, flags);
          printf("%s %s\n", opcode, operand1);
      break;
    }
    break;
  }

}

void print_alu(char *rega, char *regb, char * accumulator, char flags[])
{
  printf("Register A:  ");
  print_reg(rega);

  printf("Register B:  ");
  print_reg(regb);

  printf("Accumulator: ");
  print_reg(accumulator);

  printf("Carryflag:   %c\n", getCarryflag(flags));
  printf("Signflag:    %c\n", getSignflag(flags));
  printf("Zeroflag:    %c\n", getZeroflag(flags));
  printf("Overflowflag: %c\n", getOverflowflag(flags));
  printf("**********************\n");
}

int main()
{

  size_t nbytes = 80;
  char *cmd_line = (char *) malloc (nbytes + 1);

  /*
        read line from stdio, parse line, execute line, print result
  */

  while(getline(&cmd_line, &nbytes, stdin) != −1) {
    alu_parse_line(cmd_line);
    print_alu(rega, regb, accumulator, flags);
  }

  return 1 ;

}
```