Project 1--CDA 3101 (Spring 2014)

Worth: 100 points (10% of course grade)

Assigned: Friday, Jan 24, 2014

bits 24-22: opcode JETETE 01E9: reg A JETETE 01E6: reg B

JETBTI 015

4. LC3101 Assembly Language and Assembler (40%)

The first part of this project is to write a program to take an assembly-language program and translate it into machine language. You will

translate assembly-language names for instructions, such as beq, into their

numeric equivalent (e.g. 100), and you will translate symbolic names for addresses into numeric values. The final output will be a series of 32-bit

Hints: the example assembly-language program above is a good case to include

in your test suite, though you'll need to write more test cases to get full

credit. Remember to create some test cases that test the ability of an assembler to check for the errors in Section 4.2.

As with the assembler, you will write a suite of test cases to validate the LC3101 simulator.

The test cases for the simulator part of this project will be short assembly-

program halts. You may assume that the two input numbers are at most 15 bits

and are positive; this ensures that the (positive) result fits in an ${\tt LC3101}$

word. See the algorithm on page 252 of the

For the simulator test suite, we will correctly assemble each test case, then use it as input to a set of buggy simulators. A test case exposes a buggy simulator by causing it to generate a different answer from a correct simulator. The test suite is graded based on how many of the buggy

```
You may also choose to not use this fragment.
/* Assembler code fragment for LC3101 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define MAXIINELENGTH 1000
int readAndParse(FILE *, char *, char *, char *, char *, char *);
int isNumber(char *);
int
main(int argc, char *argv[])
    char *inFileString, *outFileString;
    FILE *inFilePtr, *outFilePtr;
    char label[MAXLINELENGTH], opcode[MAXLINELENGTH],
arg0[MAXLINELENGTH],
            arg1[MAXLINELENGTH], arg2[MAXLINELENGTH];
    if (argc != 3) {
        printf("error: usage: %s <assembly-code-file> <machine-code-</pre>
file>\n",
            arqv[0]);
        exit(1);
    }
    inFileString = argv[1];
    outFileString = argv[2];
    inFilePtr = fopen(inFileString, "r");
    if (inFilePtr == NULL) {
        printf("error in opening %s\n", inFileString);
        exit(1);
    outFilePtr = fopen(outFileString, "w");
    if (outFilePtr == NULL) {
        printf("error in opening %s\n", outFileString);
        exit(1);
    }
    /* here is an example for how to use readAndParse to read a line from
        inFilePtr */
    if (! readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2) ) {
        /* reached end of file */
    }
    /* this is how to rewind the file ptr so that you start reading from
the
        beginning of the file */
    rewind(inFilePtr);
```

```
/* after doing a readAndParse, you may want to do the following to
test the
        opcode */
    if (!strcmp(opcode, "add")) {
        /* do whatever you need to do for opcode "add" */
    }
    return(0);
}
 * Read and parse a line of the assembly-language file. Fields are
returned
 * in label, opcode, arg0, arg1, arg2 (these strings must have memory
already
 * allocated to them).
 * Return values:
       0 if reached end of file
       1 if all went well
 * exit(1) if line is too long.
 */
int
readAndParse(FILE *inFilePtr, char *label, char *opcode, char *arg0,
    char *arg1, char *arg2)
{
    char line[MAXLINELENGTH];
    char *ptr = line;
    /* delete prior values */
    label[0] = opcode[0] = arg0[0] = arg1[0] = arg2[0] = '\0';
    /* read the line from the assembly-language file */
    if (fgets(line, MAXLINELENGTH, inFilePtr) == NULL) {
     /* reached end of file */
        return(0);
    /* check for line too long (by looking for a \n) */
    if (strchr(line, '\n') == NULL) {
        /* line too long */
     printf("error: line too long\n");
     exit(1);
    /* is there a label? */
    ptr = line;
    if (sscanf(ptr, "%[^{tn}]", label)) {
     /* successfully read label; advance pointer over the label */
        ptr += strlen(label);
    }
    /*
```

```
* Parse the rest of the line. Would be nice to have real regular
              * expressions, but scanf will suffice.
           sscanf(ptr, "%*[\t\n ]%[^\t\n ]%*[\t\n ]%[^\t\n ]%[^\t\
]%*[\t\n ]%[^\t\n ]",
                      opcode, arg0, arg1, arg2);
           return(1);
}
int
isNumber(char *string)
           /* return 1 if string is a number */
           int i;
           return( (sscanf(string, "%d", &i)) == 1);
}
10. Code Fragment for Simulator
Here is some C code that may help you write the simulator. Again, you
should
take this merely as a hint. You may have to re-code this to make it do
exactly
what you want, but this should help you get started. Remember not to
change stateStruct or printState.
/* instruction-level simulator for LC3101 */
#include <stdio.h>
#include <string.h>
#define NUMMEMORY 65536 /* maximum number of words in memory */
#define NUMREGS 8 /* number of machine registers */
#define MAXLINELENGTH 1000
typedef struct stateStruct {
           int pc;
           int mem[NUMMEMORY];
           int reg[NUMREGS];
           int numMemory;
} stateType;
void printState(stateType *);
int
main(int argc, char *argv[])
           char line[MAXLINELENGTH];
           stateType state;
           FILE *filePtr;
           if (argc != 2) {
```

```
printf("error: usage: %s <machine-code file>\n", argv[0]);
     exit(1);
    filePtr = fopen(argv[1], "r");
    if (filePtr == NULL) {
     printf("error: can't open file %s", argv[1]);
     perror("fopen");
     exit(1);
    /* read in the entire machine-code file into memory */
    for (state.numMemory = 0; fgets(line, MAXLINELENGTH, filePtr) !=
NULL;
     state.numMemory++) {
     if (sscanf(line, "%d", state.mem+state.numMemory) != 1) {
          printf("error in reading address %d\n", state.numMemory);
          exit(1);
     printf("memory[%d]=%d\n", state.numMemory,
state.mem[state.numMemory]);
    return(0);
}
void
printState(stateType *statePtr)
    int i;
    printf("\n@@@\nstate:\n");
    printf("\tpc %d\n", statePtr->pc);
    printf("\tmemory:\n");
     for (i=0; i<statePtr->numMemory; i++) {
          printf("\t\tmem[ %d ] %d\n", i, statePtr->mem[i]);
    printf("\tregisters:\n");
     for (i=0; i<NUMREGS; i++) {</pre>
         printf("\t\treg[ %d ] %d\n", i, statePtr->reg[i]);
    printf("end state\n");
}
11. Programming Tips
Here are a few programming tips for writing C/C++ programs to manipulate
bits:
1) To indicate a hexadecimal constant in, precede the number by 0x. For
example, 27 decimal is 0x1b in hexadecimal.
```

2) The value of the expression (a >> b) is the number "a" shifted right

by "b"

bits. Neither a nor b are changed. E.g. (25 >> 2) is 6. Note that 25 is 11001 in

binary, and 6 is 110 in binary.

3) The value of the expression (a << b) is the number "a" shifted left by "b"

bits. Neither a nor b are changed. E.g. (25 << 2) is 100. Note that 25 is 11001

in binary, and 100 is 1100100 in binary.

4) To find the value of the expression (a & b), perform a logical AND on each

bit of a and b (i.e. bit 31 of a ANDED with bit 31 of b, bit 30 of a ANDED with

bit 30 of b, etc.). E.g. (25 & 11) is 9, since:

11001 (binary)

& 01011 (binary)

- = 01001 (binary), which is 9 decimal.
- 5) To find the value of the expression (a \mid b), perform a logical OR on each bit

of a and b (i.e. bit 31 of a ORED with bit 31 of b, bit 30 of a ORED with bit 30 $\,$

of b, etc.). E.g. (25 | 11) is 27, since:

11001 (binary) & 01011 (binary)

a 01011 (201121)

- = 11011 (binary), which is 27 decimal.
- 6) ~a is the bit-wise complement of a (a is not changed).

Use these operations to create and manipulate machine-code. E.g. to look at bit

- 3 of the variable a, you might do: (a>>3) & 0x1. To look at bits (bits 15-12) of
- a 16-bit word, you could do: (a>>12) & 0xF. To put a 6 into bits 5-3 and a 3 $\,$

into bits 2-1, you could do: $(6 << 3) \mid (3 << 1)$. If you're not sure what an operation is doing, print some intermediate results to help you debug.

12. Example Run of Simulator

memory[0] = 8454151

memory[1] = 9043971

memory[2] = 655361

memory[3]=16842754

memory[4]=16842749

memory[5] = 29360128

memory[6] = 25165824

```
memory[7]=5
memory[8]=-1
memory[9]=2
999
state:
     pc 0
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[6]25165824
           mem[ 7 ] 5
           mem[8]-1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 0
           reg[ 2 ] 0
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
000
state:
     pc 1
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 5
           reg[ 2 ] 0
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
```

```
@ @ @
state:
     pc 2
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[8]-1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 5
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     рс 3
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[8]-1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 4
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
@ @ @
state:
     pc 4
```

```
memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[8]-1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 4
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     pc 2
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[3]16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 4
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     pc 3
     memory:
           mem[ 0 ] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
```

```
mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 3
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     pc 4
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 3
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     pc 2
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
           mem[3]16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
```

```
mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
      registers:
           reg[ 0 ] 0
           reg[ 1 ] 3
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
000
state:
     рс 3
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
      registers:
           reg[ 0 ] 0
           reg[ 1 ] 2
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     pc 4
     memory:
           mem[ 0 ] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[7]5
           mem[ 8 ] -1
           mem[ 9 ] 2
      registers:
```

```
reg[ 0 ] 0
           reg[ 1 ] 2
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
000
state:
     pc 2
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 2
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     рс 3
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[7]5
           mem[8]-1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 1
           reg[ 2 ] -1
           reg[ 3 ] 0
```

```
reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
000
state:
     pc 4
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 1
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
999
state:
     pc 2
     memory:
           mem[ 0 ] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[3]16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[7]5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 1
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
```

```
end state
999
state:
     pc 3
     memory:
           mem[0] 8454151
           mem[ 1 ] 9043971
           mem[2]655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
      registers:
            reg[ 0 ] 0
            reg[ 1 ] 0
           reg[ 2 ] -1
            reg[ 3 ] 0
            reg[ 4 ] 0
            reg[ 5 ] 0
           reg[ 6 ] 0
            reg[ 7 ] 0
end state
<u>a</u> a a
state:
     pc 6
     memory:
           mem[ 0 ] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[7]5
           mem[ 8 ] -1
           mem[ 9 ] 2
      registers:
           reg[ 0 ] 0
            reg[ 1 ] 0
            reg[ 2 ] -1
            reg[ 3 ] 0
            reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
machine halted
total of 17 instructions executed
final state of machine:
```

```
000
state:
     pc 7
     memory:
           mem[ 0 ] 8454151
           mem[ 1 ] 9043971
           mem[ 2 ] 655361
           mem[ 3 ] 16842754
           mem[ 4 ] 16842749
           mem[ 5 ] 29360128
           mem[ 6 ] 25165824
           mem[ 7 ] 5
           mem[ 8 ] -1
           mem[ 9 ] 2
     registers:
           reg[ 0 ] 0
           reg[ 1 ] 0
           reg[ 2 ] -1
           reg[ 3 ] 0
           reg[ 4 ] 0
           reg[ 5 ] 0
           reg[ 6 ] 0
           reg[ 7 ] 0
end state
```