

ARM Project

...

Group 39

Tom Price, Sarthak Kumar, Shiraz Butt and Andy Hume

Overview

- Tests
- Emulator structure
- Assembler structure
- Extension
- Reflections

Tests

Emulator

Design

```
44 // initialise fetched and current instruction
45 uint32_t fetchedInstr = getNextInstr();
46 uint32_t currInstr = fetchedInstr;
47
48 // loop until next instruction is 0 (halt instruction)
49 while (currInstr) {
50     // gets next instruction and increments PC
51     currInstr = fetchedInstr;
52     fetchedInstr = getNextInstr();
53
54     // skip this instruction if condition says to
55     if (!checkCond(currInstr)) {
56         continue;
57     }
58
59     // decode and execute the instruction
60     bool wasBranch = decodeAndExecute(currInstr);
61
62     if (wasBranch) {
63         fetchedInstr = getNextInstr();
64     }
65 }
```

Good point

1. Passed all the test cases

Bad points

- Global variable for the memory and registers
 - No encapsulation
 - Difficult to debug
 - Restricted design options (couldn't put the Armstate on the stack)
- Really slow: loop01 timed out

Bad points

- Not extendable
 - When handling branch instructions we had to change main
 - Can't emulate multiple programs
- Duplicated code

Assembler

Design

Java > C

Design

Assembler 'Class'

```
22  typedef struct Assembler {
23      char *sourcePath;
24      char *binaryPath;
25      char **sourceLines;
26      int numLines;
27      uint32_t *binaryProgram;
28      int numInstrs;
29      int firstEmptyAddr;
30      int currInstrAddr;
31      ListMap *symbolTable;
32  } Assembler;
33
34  Assembler *newAssembler(char *sourcePath, char *binaryPath);
35  void assemblerInit(Assembler *this, char *sourceFile, char *binaryPath);
36  void assemble(Assembler *this);
37  void assemblerDeInit(Assembler *this);
38  void assemblerDeconstruct(Assembler *this);
```

Design

Assemble Function

```
58
59 // assembles the source file
60 ⇐ void assemble(Assembler *this) {
61     // do first pass
62     createSymbolTableAndCountInstrs(this);
63
64     // do second pass
65     parseInstructions(this);
66
67     // write to binary file
68     writeToBinaryFile(this);
69     }
```

Design

- Token handlers - 👍
 - Gives opportunity to greatly simplify some instructions (like halt)

```
108  
109 ↩ uint32_t handleHalt(Assembler *assembler, char **tokens) {  
110     return 0;  
111     }  
112
```

- Map for the symbol table and function lookup
 - Use void pointers to make map generic - some complications
 - ISO C forbids casting from void pointer to function pointer
 - Hacky workaround - void ** to function pointer pointer, then dereference



First Pass

1. Count number of lines in the source file for source lines array, read them into it
2. Iterate through lines
 - a. Strip leading space and comments, skip if line is empty
 - b. Add labels to symbol table
 - c. Count number of instructions while going

Second Pass

Iterate through the source lines array:

1. Break line into tokens
2. Pass to appropriate handling function based on mnemonic using function table
 - a. Use common components (operand2 for DP and address for SDT very similar) to work out fields
 - b. If we need to add a constant to the end of the file, use and update nextFreeAddr variable, calculated based on the number of instructions present in the source file
 - c. Pass the fields to instruction generators that make the 32 bit instruction
3. Write all instructions out to the file when done

Extension

Compiler?

3 bit counter

Reflections