PMAT 402 - Systems Programming
Assignment -2
SIC/XE Instruction Parser

Gandholi Sarat - 23008

April 10, 2025

Contents

I Objective

The objective of this program is to implement a C++ class that parses SIC (Simplified Instructional Computer) instructions and extracts key components from them. The program takes a hexadecimal SIC instruction (either 6 or 8 digits) as input and performs the following tasks:

- **Fetch Opcode:** Extract the 6-bit operation code using the `getOpcode()` function.
- **Extract Flags (nixbpe):** Retrieve the six control bits using the `getFlags()` function, which represent addressing and format options.
- **Determine Addressing Mode:** Analyze the flags to identify whether the instruction uses immediate, indirect, indexed, base-relative, PC-relative, or direct addressing via the `getAddressingMode()` function.
- **Compute Displacement Address:** Calculate the memory address involved in the instruction using the `getDispAddr()` function.
- **Identify Instruction Format:** Determine whether the instruction uses Format 3 or Format 4 by examining the `e` flag via the `getFormat()` function.

II Code Listing

```
#include <iostream>
#include <bitset>
#include <string>
#include <sstream>
#include <iomanip>
#include <cstdint>

class MyParse {
    public:  // Make the function public
        // Convert hex string to 32-bit unsigned integer
        uint32_t hexTo32BitUnsigned(const std::string& hexInput) {
            uint32_t value;
            std::stringstream ss;
            ss << std::hex << hexInput;
            ss >> value;

            if (hexInput.length() == 6) {
                // Append zeros by shifting left 8 bits
                uint32_t value32Bit = value << 8;
                return value32Bit;
            }
            else if (hexInput.length() == 8) {
                // Already a 32-bit value
                return value;
            }
            else {
```

```
27                  // Invalid input length
28                  std::cerr << "Invalid input: Please provide either 6
    or 8 hexadecimal digits." << std::endl;
29                  return 0;  // Returning 0 for invalid inputs
30              }
31          }
32
33          // Extract opcode (6 bits) from binary instruction
34          std::bitset<6> getOpcode(uint32_t binary){
35              uint32_t opcode = (binary >> 26) & 0x3F;
36              return std::bitset<6> (opcode);
37          }
38
39          // Extract flags (nixbpe) from binary instruction
40          std::bitset<6> getFlags (uint32_t binary){
41              uint32_t flags = (binary >> 20) & 0x3F;
42              return std::bitset<6> (flags);
43          }
44
45          // Determine addressing mode based on flags
46          std::string getAddressingMode(std::bitset<6> flags){
47              if (!flags[5] && flags[4] && !flags[3] )
48                  return "Immediate Addressing Mode";
49              if (flags[5] && !flags[4] && !flags[3] )
50                  return "Indirect Addressing Mode";
51              if (flags[3] && !flags[2] && !flags[1] )
52                  return "Index Addressing Mode";
53              if (flags[2] && !flags[1] )
54                  return "Base Relative Addressing Mode";
55              if (!flags[2] && flags[1] )
56                  return "Program-Counter Relative Addressing Mode";
57              if (!flags[2] && !flags[1] )
58                  return "Direct Addressing Mode";
59              else
60                  return "Unknown Addressing Mode";
61          }
62
63          // Compute Displacement address based on flags and hex input
64          std::bitset<20> getDispAddr(const std::string& hexInput) {
65              uint32_t value = hexTo32BitUnsigned(hexInput);
66              auto flags = getFlags(value);
67
68              uint32_t DisplacementAddress = 0;
69
70              if (flags[0]) { // Format 4: Use 20-bit Displacement
    address
71                  DisplacementAddress = value & 0xFFFFF;  // Mask to get
     the lower 20 bits
72              }
73              else { // Format 3: Use 12-bit Displacement address
74                  DisplacementAddress = value & 0xFFFFF;  // Mask to get
     the lower 12 bits
75              }
76
```

```
77                  // Return the Displacement address as a 20-bit bitset
78                  return std::bitset<20>(DisplacementAddress);
79              }
80
81              // Determine instruction format (3 or 4) based on flags
82              unsigned int getFormat(std::bitset<6> flags) {
83                  return flags[0] ? 4 : 3;  // Format 4 if e-bit is set,
    else Format 3
84              }
85      };
86
87      int main() {
88          MyParse parser;
89
90          // Take input from user
91          std::string hexInput;
92          std::cout << "Enter a hexadecimal instruction (6 or 8 characters):
     ";
93          std::cin >> hexInput;
94
95          // Validate input length
96          if (hexInput.length() != 6 && hexInput.length() != 8) {
97              std::cerr << "Error: Invalid input length. Please provide
    either 6 or 8 hexadecimal digits." << std::endl;
98              return 1; // Exit with error code
99          }
100
101         // Convert hex input to binary
102         uint32_t binary = parser.hexTo32BitUnsigned(hexInput);
103
104         // Extract and display opcode
105         auto opcode = parser.getOpcode(binary);
106         std::cout << "Opcode: " << opcode << "\n";
107
108         // Extract and display flags (nixbpe)
109         auto flags = parser.getFlags(binary);
110         std::cout << "Flags (nixbpe): " << flags << "\n";
111
112         // Determine and display addressing mode
113         auto addressingMode = parser.getAddressingMode(flags);
114         std::cout << "Addressing Mode: " << addressingMode << "\n";
115
116         // Compute and display display address
117         auto DisplacementAddress = parser.getDispAddr(hexInput);
118         std::cout << "Disp Address: " << DisplacementAddress << "\n";
119
120         // Determine and display instruction format
121         auto format = parser.getFormat(flags);
122         std::cout << "Instruction Format: " << format << "\n";
123
124         return 0;
125     }
```

Listing 1: SIC Instruction Parser in C++

III Code Output

```
Enter a hexadecimal instruction (6 or 8 characters): 032600
Opcode: 000000
Flags (nixbpe): 110010
Addressing Mode: Program-Counter Relative Addressing Mode
Disp Address: 01100000000000000000
Instruction Format: 3

Enter a hexadecimal instruction (6 or 8 characters): 03C300
Opcode: 000000
Flags (nixbpe): 111100
Addressing Mode: Base Relative Addressing Mode
Disp Address: 00110000000000000000
Instruction Format: 3

Enter a hexadecimal instruction (6 or 8 characters): 022030
Opcode: 000000
Flags (nixbpe): 100010
Addressing Mode: Indirect Addressing Mode
Disp Address: 00000011000000000000
Instruction Format: 3

Enter a hexadecimal instruction (6 or 8 characters): 010030
Opcode: 000000
Flags (nixbpe): 010000
Addressing Mode: Immediate Addressing Mode
Disp Address: 00000011000000000000
Instruction Format: 3

Enter a hexadecimal instruction (6 or 8 characters): 003600
Opcode: 000000
Flags (nixbpe): 000011
Addressing Mode: Program-Counter Relative Addressing Mode
Disp Address: 01100000000000000000
Instruction Format: 4

Enter a hexadecimal instruction (6 or 8 characters): 0310C303
Opcode: 000000
Flags (nixbpe): 110001
Addressing Mode: Direct Addressing Mode
Disp Address: 00001100001100000011
Instruction Format: 4
```

IV Output comparison

	op	n	i	x	b	p	e	disp/address
032600	000000	1	1	0	0	1	0	0110 0000 0000
03C300	000000	1	1	1	1	0	0	0011 0000 0000
022030	000000	1	0	0	0	1	0	0000 0011 0000
010030	000000	0	1	0	0	0	0	0000 0011 0000
003600	000000	0	0	0	0	1	1	0110 0000 0000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011

Figure 1: Expected ouput, Table from textbook

The output of the program is consistent with the expected results from textbook for various SIC instructions. The opcode, flags, addressing mode, displacement address, and instruction format are accurately extracted and displayed based on the provided hexadecimal input.

V Function Descriptions

V.I hexTo32BitUnsigned

Converts a 6- or 8-digit hexadecimal string into a 32-bit unsigned integer. If the input is 6 digits, it shifts the value left by 8 bits to simulate 24-bit instructions.

V.II getOpcode

Extracts the first 6 bits from the most significant end of the instruction to retrieve the opcode.

V.III getFlags

Extracts the `nixbpe` flags from bits 20-25 of the 32-bit instruction.

V.IV getAddressingMode

Determines the addressing mode using the flags:

- **Immediate:** `n=0, i=1`
- **Indirect:** `n=1, i=0`
- **Index:** `x=1`
- **Base-relative:** `b=1, p=0`
- **PC-relative:** `b=0, p=1`
- **Direct:** `b=0, p=0`

V.V getDispAddr

Computes the Displacement address:

- For Format 4 (e=1), uses a 20-bit address.
- For Format 3 (e=0), masks similarly to maintain 12-bit Displacement addressing.

V.VI getFormat

Returns the instruction format:

- Format 3: if `e = 0`
- Format 4: if `e = 1`