Matt Bennett

ELEC 3275 - Computer Architecture
Assignment 6 - ARM Decoder

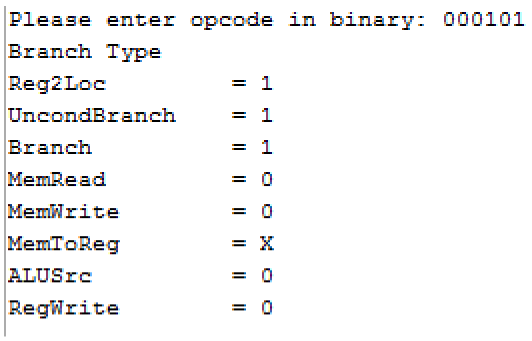
Description

To address the problem of creating a ARMv8 decoder/control unit for the provided instructions, the instructions, the corresponding opcode, and the instruction format were combined in a spreadsheet. The resulting table was grouped by format and each instruction . Once the instructions were grouped, the control bits each format required were determined using the provided processor diagram and recorded in the table. Grouping the opcode instructions by format, a pattern was observed in the 4th to 6th bits of the opcode. There are five exceptions to the pattern that were noted, STXR, LDXR, BR, LSR, and LSL. The BR instruction which is considered a register format instruction resulting in a branching format in this pattern is logical given that it corresponds to the branch to register instruction. Likewise, the LSR and LSL, logical shift instructions are considered register format, however, both are considered immediate types in the observed pattern. Considering logical shifts as immediate types is consistent with both instructions use of the ALU. The STXR and LDXR instructions are data transfer format instructions and are seen as register type using this pattern. An explanation for the STXR and LDXR exceptions was not found.

The pattern used is a simple pattern of if-else statements that first determines if the format is branching, immediate, register, or data formats using the 4-6 bit patterns shown. If the opcode is determined to correspond to a branching instructions, the second two bits are then used to determine if the branch type is conditional or unconditional. If the instruction is determined to correspond to a immediate format, the 9th bit is used to determine if an immediate or an immediate move instruction was entered. Finally, if the instruction was determined to be a data type, the 9th and 10th bit are used to determine if the instruction is to load or store data.

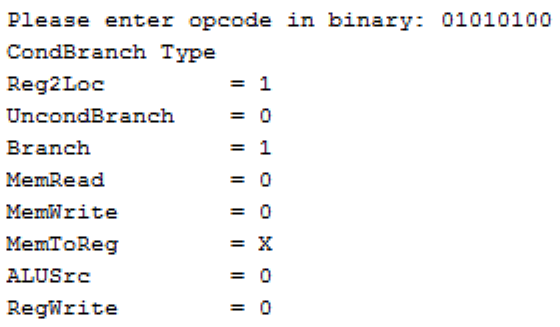
Instruction Format	Bits 4 to 6
Branch	101
Conditional Branch	101
Data Transfer	110
Immediate	100
Move	100
Register	010

Results



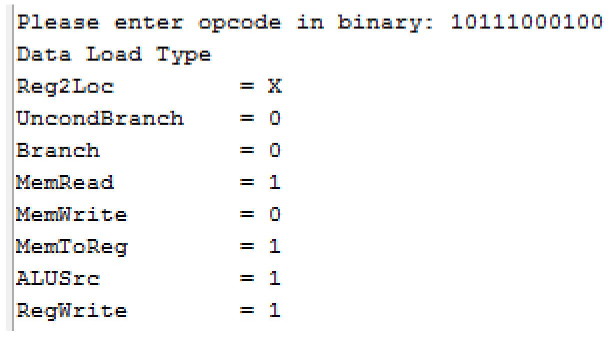
```
Please enter opcode in binary: 000101
Branch Type
Reg2Loc         = 1
UncondBranch    = 1
Branch          = 1
MemRead         = 0
MemWrite        = 0
MemToReg        = X
ALUSrc          = 0
RegWrite        = 0
```

B - branch test



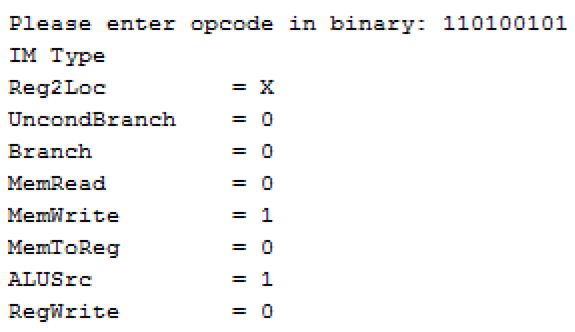
```
Please enter opcode in binary: 01010100
CondBranch Type
Reg2Loc         = 1
UncondBranch    = 0
Branch          = 1
MemRead         = 0
MemWrite        = 0
MemToReg        = X
ALUSrc          = 0
RegWrite        = 0
```

B.cond - conditional branch test



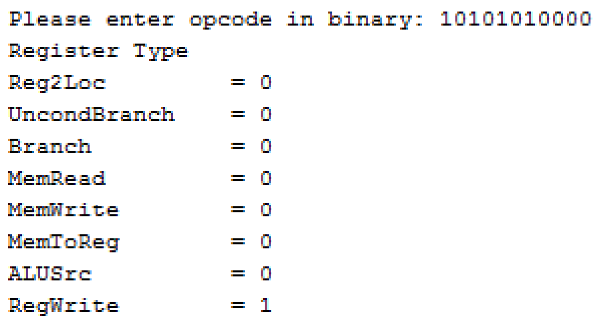
```
Please enter opcode in binary: 10111000100
Data Load Type
Reg2Loc         = X
UncondBranch    = 0
Branch          = 0
MemRead         = 1
MemWrite        = 0
MemToReg        = 1
ALUSrc          = 1
RegWrite        = 1
```

LDURSW - data transfer test



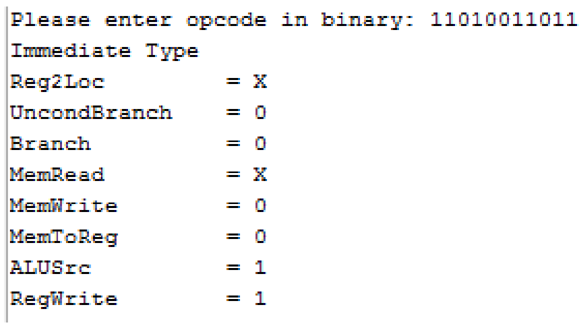
```
Please enter opcode in binary: 110100101
IM Type
Reg2Loc         = X
UncondBranch    = 0
Branch          = 0
MemRead         = 0
MemWrite        = 1
MemToReg        = 0
ALUSrc          = 1
RegWrite        = 0
```

MOVZ - immediate move test



```
Please enter opcode in binary: 10101010000
Register Type
Reg2Loc         = 0
UncondBranch    = 0
Branch          = 0
MemRead         = 0
MemWrite        = 0
MemToReg        = 0
ALUSrc          = 0
RegWrite        = 1
```

ORR - register test



```
Please enter opcode in binary: 11010011011
Immediate Type
Reg2Loc         = X
UncondBranch    = 0
Branch          = 0
MemRead         = X
MemWrite        = 0
MemToReg        = 0
ALUSrc          = 1
RegWrite        = 1
```

LSL - non pattern consistent result

Code

```
/*
 * File:   main.c
 * Authors: Adam Ziel, Matt Bennett
 *
 * Created on June 26, 2016, 12:24 PM
 *
 * ELEC3725 Assignment #6 - ARMv8 Decoder
 */

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

/*
 * Determine control bits for an ARMv8 binary opcode stored as ASCII characters
 * of 1's and 0's with index 0 being the largest place value.
 *
 * Return 9 element character array with i[0] being a letter to represent the
 * format of the opcode and i[1:8] representing the needed control bit value.
 */
char* decode( char * inputBinary ){
    char *control = malloc(12);
    if ( inputBinary[ 5 ] == '1' ) {
       if ( inputBinary[ 1 ] == '0' && inputBinary[ 2 ] == '0' ){
            // BR is decoded as branch type and not register type.
            // Branch Type
            control = "B11100X00";
        } else {
            // Conditional Branch Type
            control = "C10100X00";
        }
    } else if ( inputBinary[ 4 ] == '0' ){
        if ( inputBinary[ 8 ] == '0' ){
            // LSL & LSR are decoded as immediate types and not data types.
            // Immediate Type
            control = "IX00X0011";
        } else{
            // IM Type
            control = "MX0001010";
        }
    } else if ( inputBinary[ 3 ] == '0' ){
        // STXR & LDXR are decoded as register types and not data types.
        // Register Type
        control = "R00000001";
    } else{
        // STXR & LDXR are decoded as register types and not data types.
        // BR, LSL, & LSR are decoded as branch, immediate, & immediate types
        // and not data types.
        if ( inputBinary[ 8 ] == '0' && inputBinary[ 9 ] == '0' ){
            // Data Type - Store
            control = "S10001X10";
        } else{
            // Data Type - Load
            control = "LX0010111";
        }
    }
    return control;
}


/*
 * Receive console input from user corresponding to ARMv8 binary opcode.
 * Determine the resulting control bits needed for the provided opcode.
 * Provide basic results to console.
 */
int main(int argc, char** argv) {
    while ( 1 ) {
        char inputBinary[ 11 ];
        printf( "Please enter opcode in binary: " );
        scanf( "%s", &inputBinary );
        getchar();

        char *control = decode( inputBinary );
        char *type = malloc( 10 );
        switch ( control[ 0 ] ){
            case 'B': type = "Branch"; break;
            case 'C': type = "CondBranch"; break;
            case 'I': type = "Immediate"; break;
            case 'M': type = "IM"; break;
            case 'R': type = "Register"; break;
            case 'S': type = "Data Store"; break;
            case 'L': type = "Data Load"; break;
        }
        printf( "%s Type\n", type );

        const char *nameList[ 8 ] = { "Reg2Loc", "UncondBranch", "Branch",
                "MemRead", "MemWrite", "MemToReg", "ALUSrc", "RegWrite" };
        for ( char i = 0; i < 8; i++ ){
            printf( "%-15s", nameList[ i ]);
            printf( " = %c\n", control[ i+1 ] );
        }

        char continueChar[ 1 ];
        printf( "\nContinue? (Y/N) " );
        scanf( "%s", &continueChar );
        getchar();
        if ( *continueChar != 'Y' && *continueChar != 'y' ){
            break;
        }
    }
    return (EXIT_SUCCESS);
}
```