ARM Simulator

CSE-112 Project

OBJECTIVE

To design and build a function simulator for a subset of ARM assembly instructions in a high level language.

TEAM MEMBERS

➔ Lavina Jain	(2016052)
➔ Siddharth Dhawan	(2016096)
➔ Kanika Saini	(2016047)

LANGUAGE

C++

METHODOLOGY

MEMORY

The instruction memory and the data memory (for store operations) are modelled as an **INS.MEM** file and a **DATA.MEM** file respectively.

REGISTERS

The registers are represented by an array of *long* (32-bit integer) - **R[16]**.

CPSR FLAGS

C, V, Z, and **N** flags are represented by **global boolean variables**.

INSTRUCTIONS

The current decoded instruction is stored as a global object of the class **Instruction**.

Every supported instruction is represented by its own class which inherits from the superclass **Instruction** and has the following three functions : -

- ➔ execute()
  - ◆ Execute the operation corresponding to the *Execute* step of that specific instruction . For example: Execute operation for Add class involves adding the two operands

- ➔ memory()
  - ◆ Read/Write to the required memory location. For example: Reading/Writing to memory in case of STR/LDR instructions.

- ➔ writeBack()
  - ◆ Store some value in the destination register. For example: The Writeback step in Add instruction involves writing to the destination register the result of addition operation.

SIMULATOR

The Simulator functionality is performed in the main() function of the **ARMSIM.cpp** file. The file contains methods for each of Fetch, Decode, Execute, Memory and WriteBack stages of the functioning of an ARM processor.

- ➔ fetch()
  - ◆ Reads a single line from the INS.MEM file as *string*.
  - ◆ Separate the instruction code and address and display.

- ➔ decode()
  - ◆ Converts the instruction hex code to binary code stored in a C++ *bitset*<>
  - ◆ Decode the binary code to separate the operation *string* and the registers.
  - ◆ Read the values at the registers.
  - ◆ Create an object of the Instruction Class and store it as the current decoded instruction

- ➔ execute()
  - ◆ Call the execute() function of the current decoded instruction.

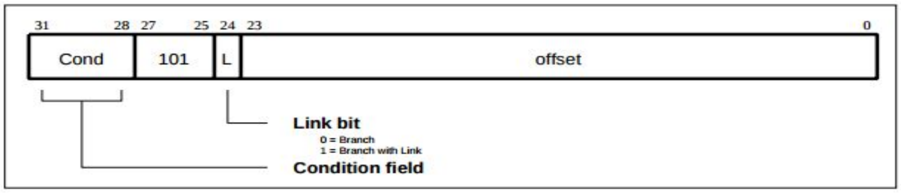
- ➔ memory()
  - ◆ Call the memory() function of the current decoded instruction.

- ➔ writeBack()
  - ◆ Call the writeBack() function of the current decoded instruction.

FEATURES SUPPORTED

INSTRUCTIONS

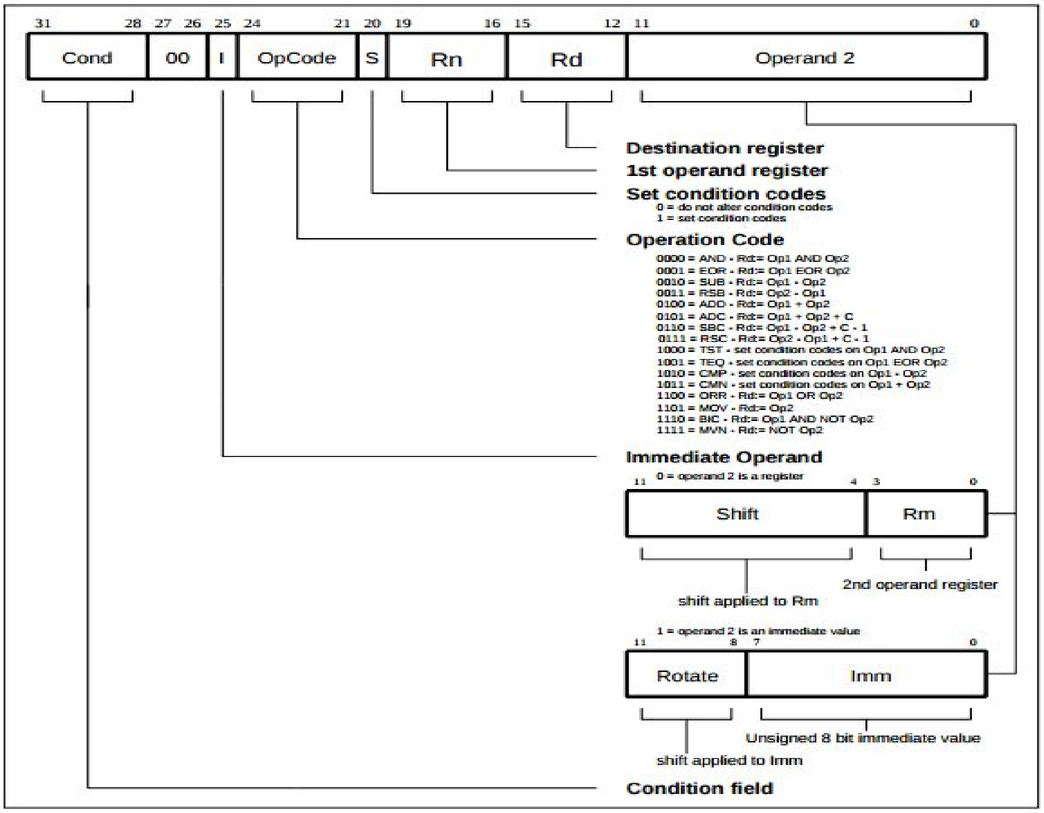
- Branch Instruction - B



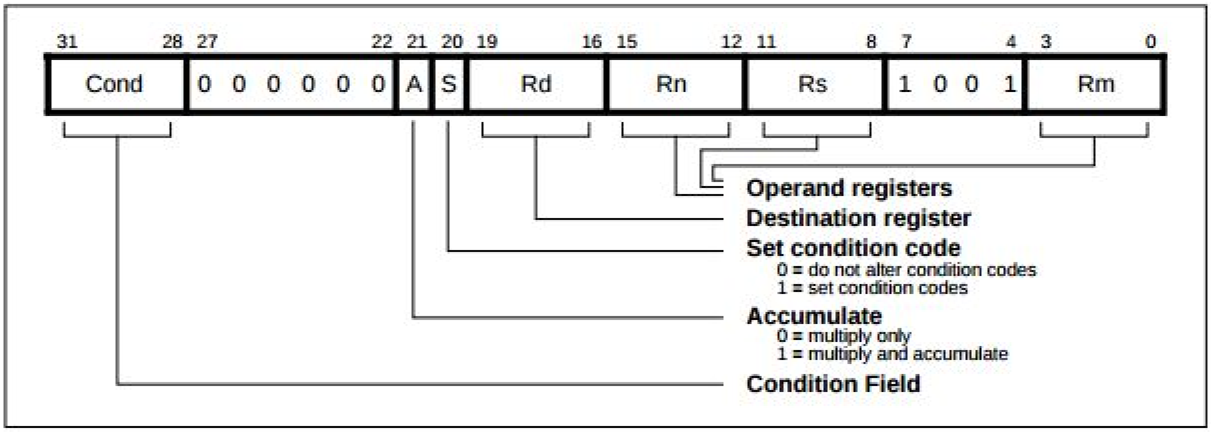
Condition Codes

Suffix	Flags	Meaning
EQ	Z set	equal
NE	Z clear	not equal
GE	N equals V	greater or equal
LT	N not equal to V	less than
GT	Z clear AND (N equals V)	greater than
LE	Z set OR (N not equal to V)	lesser or equal
AL	ignored	always

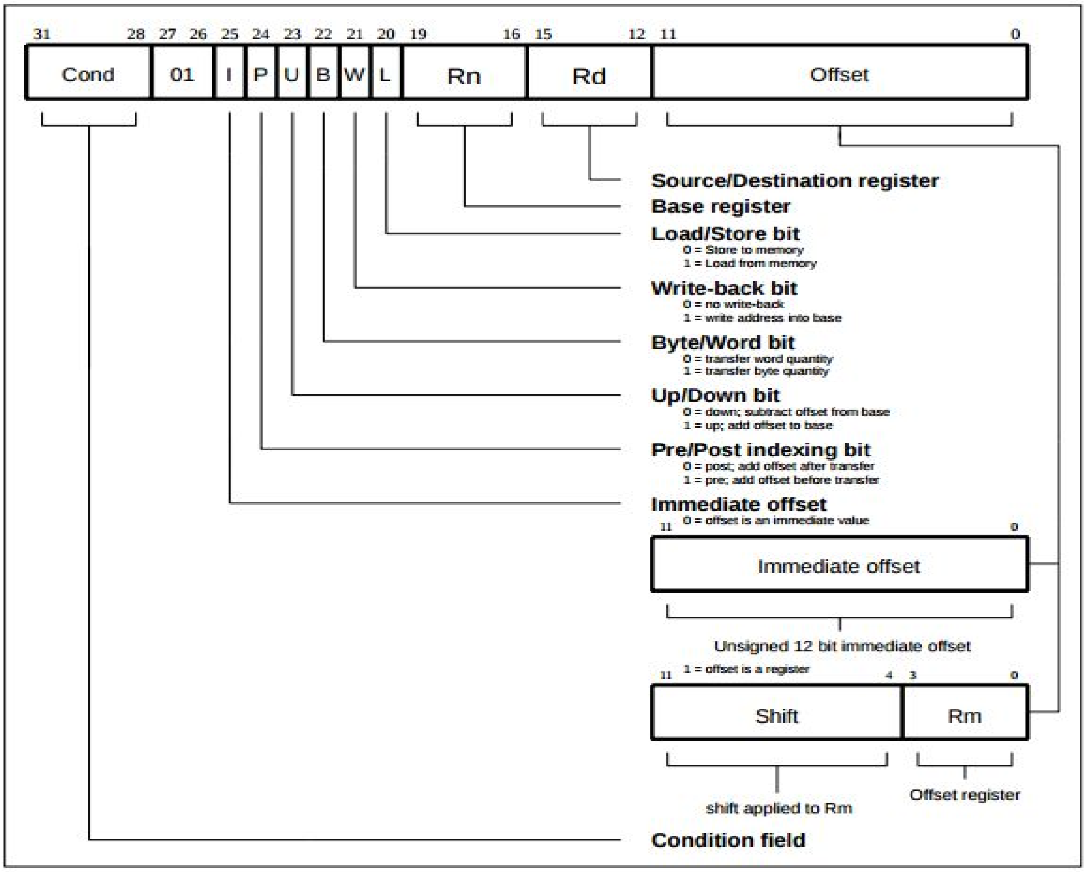
- Data Processing Instructions
    - AND
    - ORR
    - EOR
    - MOV
    - MVN
    - ADD
    - SUB
    - CMP


31
28
27
26
25
24
21
20
19
16
15
12
11
0
Cond
00
I
OpCode
S
Rn
Rd
Operand 2
Destination register
1st operand register
Set condition codes
0 = do not alter condition codes
1 = set condition codes
Operation Code
0000 = AND - Rd:= Op1 AND Op2
0001 = EOR - Rd:= Op1 EOR Op2
0010 = SUB - Rd:= Op1 - Op2
0011 = RSB - Rd:= Op2 - Op1
0100 = ADD - Rd:= Op1 + Op2
0101 = ADC - Rd:= Op1 + Op2 + C
0110 = SBC - Rd:= Op1 - Op2 + C - 1
0111 = RSC - Rd:= Op2 - Op1 + C - 1
1000 = TST - set condition codes on Op1 AND Op2
1001 = TEQ - set condition codes on Op1 EOR Op2
1010 = CMP - set condition codes on Op1 - Op2
1011 = CMN - set condition codes on Op1 + Op2
1100 = ORR - Rd:= Op1 OR Op2
1101 = MOV - Rd:= Op2
1110 = BIC - Rd:= Op1 AND NOT Op2
1111 = MVN - Rd:= NOT Op2
Immediate Operand
0 = operand 2 is a register
11
4
3
0
Shift
Rm
2nd operand register
shift applied to Rm
1 = operand 2 is an immediate value
11
8
7
0
Rotate
Imm
Unsigned 8 bit immediate value
shift applied to Imm
Condition field

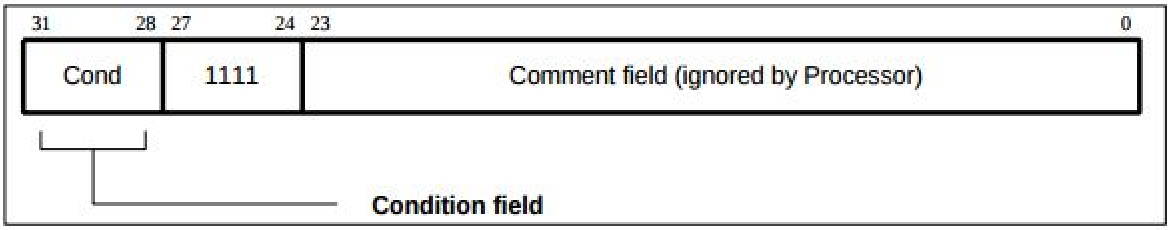
- Multiply Instructions - MUL



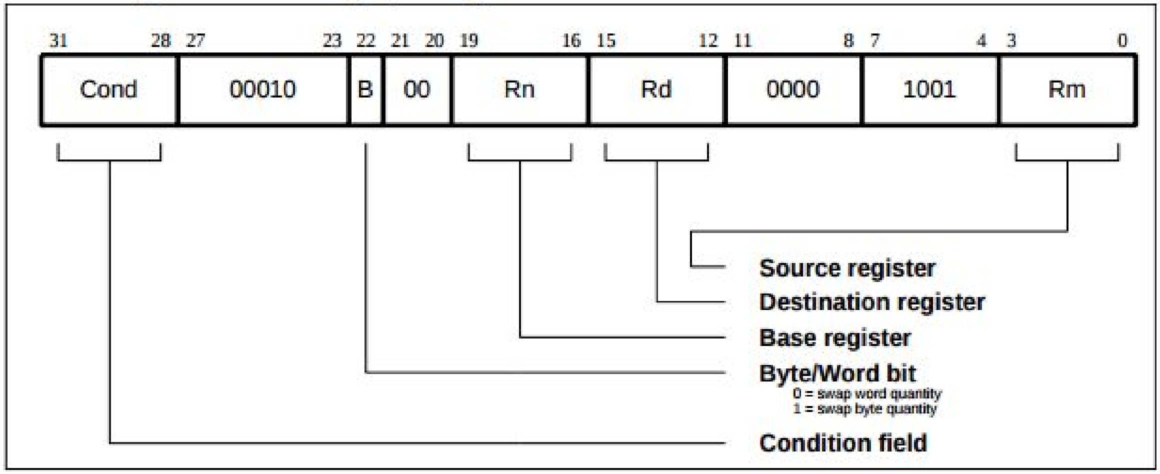
- Single Data Transfer Instructions
    - LDR
    - STR



- Software Interrupt Instruction - SWI



- Single Data Swap (SWP)


31
28
27
23
22
21
20
19
16
15
12
11
8
7
4
3
0
Cond
00010
B
00
Rn
Rd
0000
1001
Rm
Source register
Destination register
Base register
Byte/Word bit
0 = swap word quantity
1 = swap byte quantity
Condition field