COMPUTER ORGANISATION PROJECT

TEAM MEMBERS

- Ayushi Srivastava (2016025)
- Raghav Sood (2016259)
- Surabhi S Nath (2016271)

CHOSEN PROJECT TOPIC

Project Topic 1: ARM Simulator

Design and implement the function simulator in Java for subset of ARM instructions

EXTENSION OF MIDWAY CHECKPOINT

THEORY

Instruction Cycle

For each instruction the sequence of operations to be performed can be divided into several phases. Each instruction must go through the following 5 phases:

1. Fetch
2. Decode
3. Execute
4. Memory
5. Writeback

These steps together complete one instruction cycle.

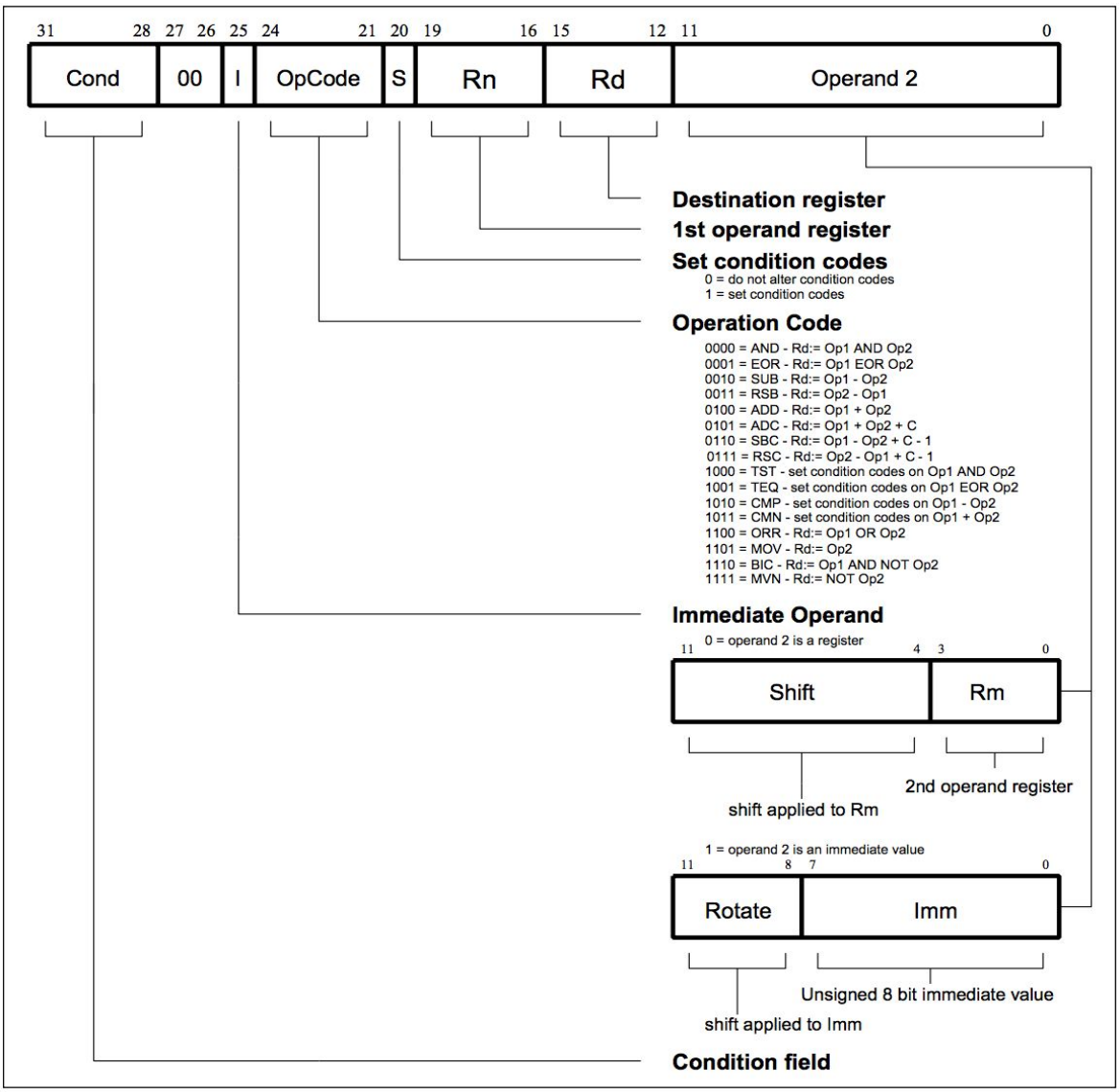
For a given instruction the 5 operations are executed sequentially, however different operations of different instructions can run parallely for faster overall execution. Let's have a look at these 5 phases in more detail:

1. **Fetch**
   The memory address is read from the PC and the instruction is fetched from memory. This instruction is stored in the Instruction Register (IR)
2. **Decode**
   The Opcode decides which command the instruction executes. For arithmetic and logical operations, the registers are read and values are fetched. In case of memory operations, the effective address is calculated
3. **Execute**
   The arithmetic and logical operations are evaluated in the execute stage
4. **Memory**
   Memory is accessed in this stage by load, store instructions
5. **Writeback**
   The registers are updated in this stage with new values based on previous stages

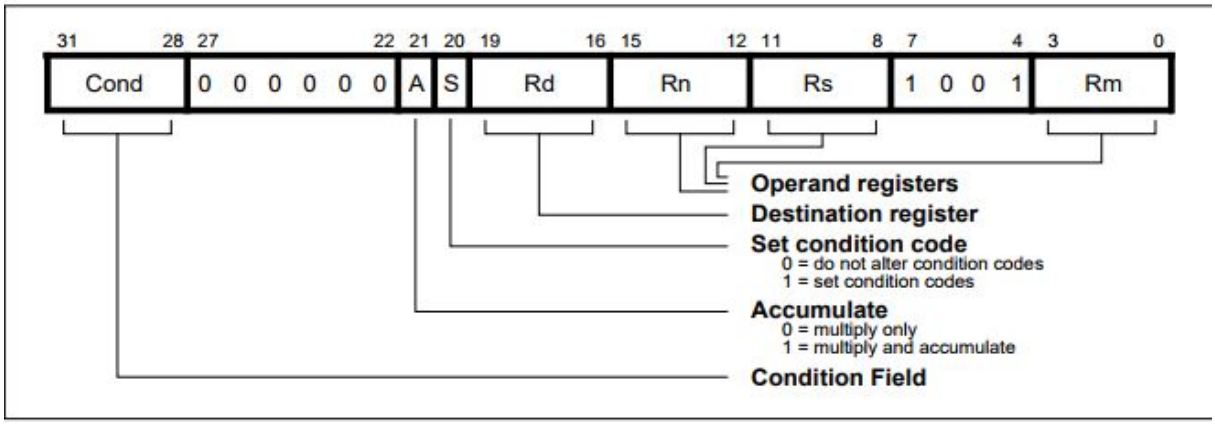
Instruction Formats

ARM instruction formats are as follows:
The first 4 bits (31,30,29,28) are for the **condition.**
Bits 27,26 indicate the **format** of instruction.

If format bits are 00, it denotes a **data processing instruction**.
Here, the 25th bit indicates if the instruction has an **immediate operand**, based on this bit, the operand 2 has a corresponding format. Also, the bits 24,23,22,21 are reserved for the opcode.



Multiply Instruction

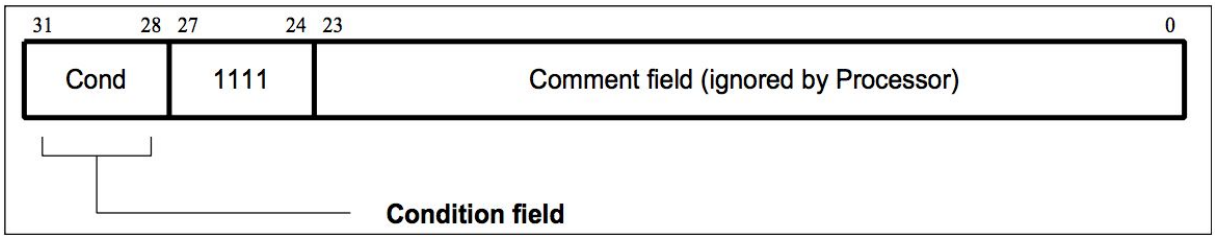


Branch Instruction

If the 27th , 26th, 25th bits are 101, it denotes a **branch instruction.**
Here, the 24th bit is the link bit. If it is 1, then its a **branch with link** instruction.

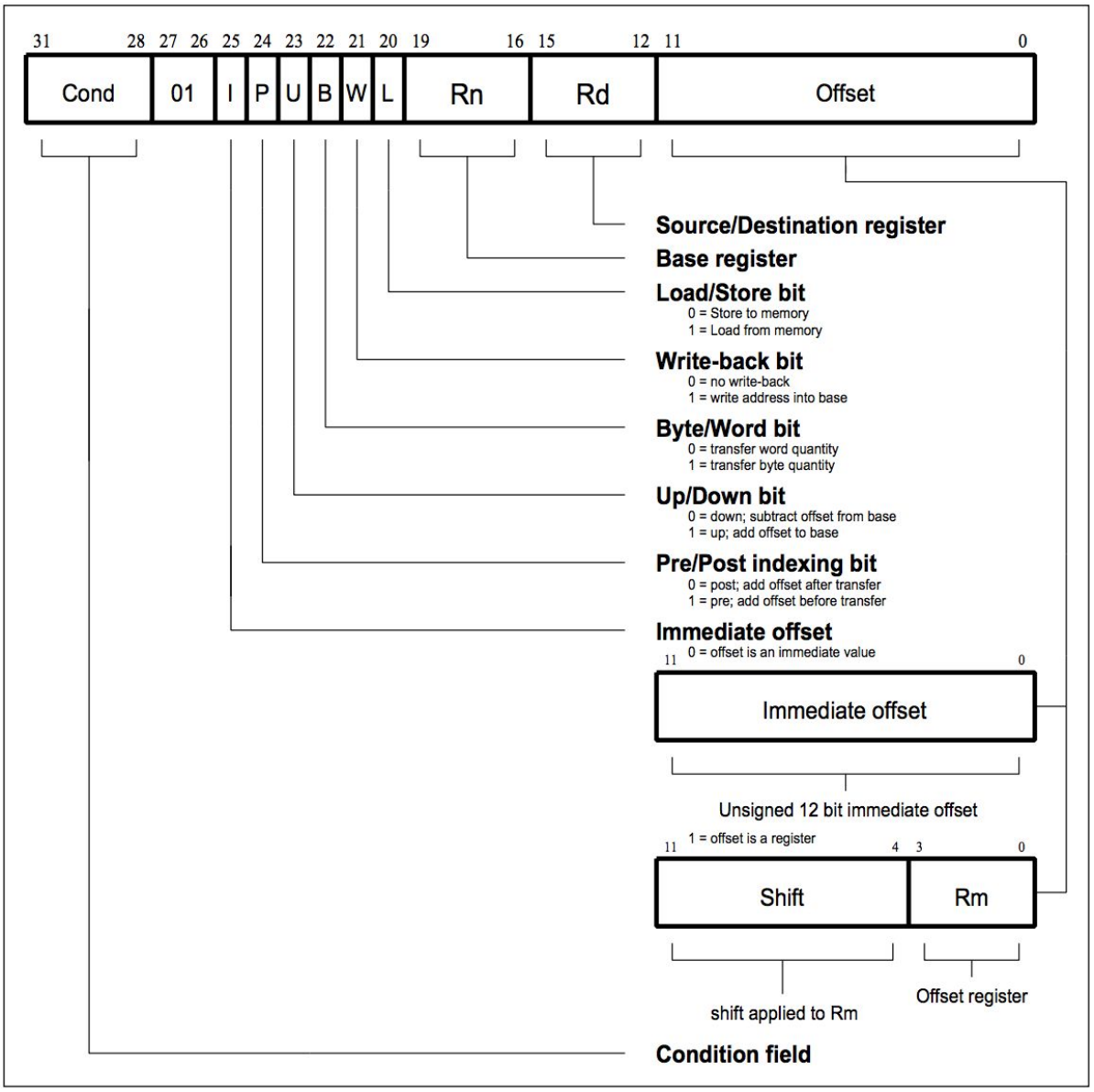
SWI instruction

If bits 27,26,25,24 are all 1, it denotes a **software interrupt instruction,** eg: print, exit



Load/Store

If the 27th, 26th bits are 01, it denotes a **load/store instruction**.
If the 20th bit is a 1, it denotes a load, if it is a 0, it denotes a store instruction.
Again, the 25th bit indicates if the instruction has an **immediate operand,** based on this which, the operand 2 has a corresponding format.



In both load store and data processing, bits 19,18,17,16 denote Base register, and bits 15,14,13,12 denote Source/Destination register.

IMPLEMENTATION

The steps we followed were as follows:

- ➔ Read the input from the provided .MEM file using FileInputStream, Reader class
- ➔ Parsed the string to separate address and instruction
- ➔ Converted address and instruction to binary
- ➔ Created two hashmaps, one mapping address to the instruction in hex and one mapping the address to the instruction in binary
- ➔ Created an array of registers
- ➔ Kept a Program Counter which is represented by R15, which increments value by 4 after each instruction
- ➔ Each instruction was assigned an ID as follows:
  String instructions[] = {"ADD", "SUB", "RSB", "MUL", "AND", "ORR", "EOR", "MOV", "MVN", "CMP", "LDR", "STR", "B", "BNE", "BL", "PRINT", "EXIT"}; where each instruction's id is the index of array. This allowed us to identify the operation to perform using the id.
- ➔ Started execution of instruction by **fetching** the first instruction
- ➔ **Decoded** the instruction by reading the bits 27, 26 to decided the instruction format
  - ◆ If they were 00, it denoted a data processing instruction which included arithmetic and logical instructions such as ADD, SUB, MUL, CMP, MOV etc. Further by reading the OPCode (bits 24-21), got the corresponding command to be executed. This was followed by reading the registers (bits 19-16 and 15-12) to obtain the values or obtain directly the immediate values to perform the operation on.
  - ◆ If they were 01, it denoted a load/store instruction such as LDR, STR. Further by reading the 20th bit, concluded whether it was a load instruction or store instruction. Here, a memory location was obtained by reading the value in base register and getting the offset value. In case of a load, the data was loaded into the destination register from this memory location and in case of store, the value in the source register was stored into this memory location.
  - ◆ If they were 10, it denoted a branch instruction. Here, the 24th bit indicated whether it was a branch, or a branch with link instruction. After this, we got the location to branch by reading the bits 23-0.
- ➔ **Executed** the data processing commands by performing the function of the arithmetic or logical instruction and calculated the results, stored them in temporary variables.

- ➔ **Memory** Load, Store instructions were needed to access the memory. Loaded and stored data from or into the memory array.
- ➔ **Writeback** was based on the results of execute and load, accordingly we updated the array of registers and printed the updated register values.
- ➔ The above steps were repeated for the next instruction
- ➔ Continued till “0xEF000011”

ISSUES FACED

- Cases where operand 2 was an immediate value or a register had to be dealt separately for every instruction
- Load/store instructions of the form: ldr r2, [r0, r1], that is r2 ← *(r0 + r1) could not be implemented because we did not know to which address the sum r0 + r1 pointed
- Similar formats for instructions eg MUL and AND needed more condition evaluations for distinguishing between them
- Product of multiply can be larger than 32 bits, so value would need to be stored in 2 separate registers

CONCLUSION AND FUTURE WORK

The set objectives of the project were successfully met as per the proposed plan and indicated timeline. The ARM instructions were simulated and the details of the instruction cycle stages were displayed. However, we dealt with only the basic ARM instructions in this project which can be extended to more advanced commands such as implementing recursive procedures as future work. The project led to a good understanding of ARM and low level languages and was an enjoyable learning experience.