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

METHODOLOGY AND PROJECT PLAN

- ### Understanding of the Project Topic

Introduction

A computer can only understand binary language. So how are complex codes written by programmers understood by the computer system? The codes written in high level languages (like Java, C++) are first converted into low level assembly languages (like MIPS, ARM, x86) by the compiler, which is in turn converted to a sequence of zeroes and ones by the assembler. This sequence is then read and processed by our computer.

ARM is a RISC architecture based low level language.
RISC architecture has an emphasis on software rather than hardware, it uses simple instructions which can be processed in one clock cycle. Operations in RISC instruction set are register-to-register based since is accesses operands from the register file. RISC having same length of instructions allows overlapping of instructions in a pipelined manner to which enhances performance.

An ARM Simulator permits us to simulate the execution of ARM instructions.

Registers are small and fast memory storage elements which are located nearest to the processor and come at the top of the memory hierarchy. ARM registers are 32 bits long. It has 13 general purpose registers R0 to R12, and 3 special purpose registers namely Stack Pointer (R13) which points to the last value written onto the stack, Link Register (R14) which holds the return address of a function call and Program Counter (R15) which points to the next instruction to be executed.

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.

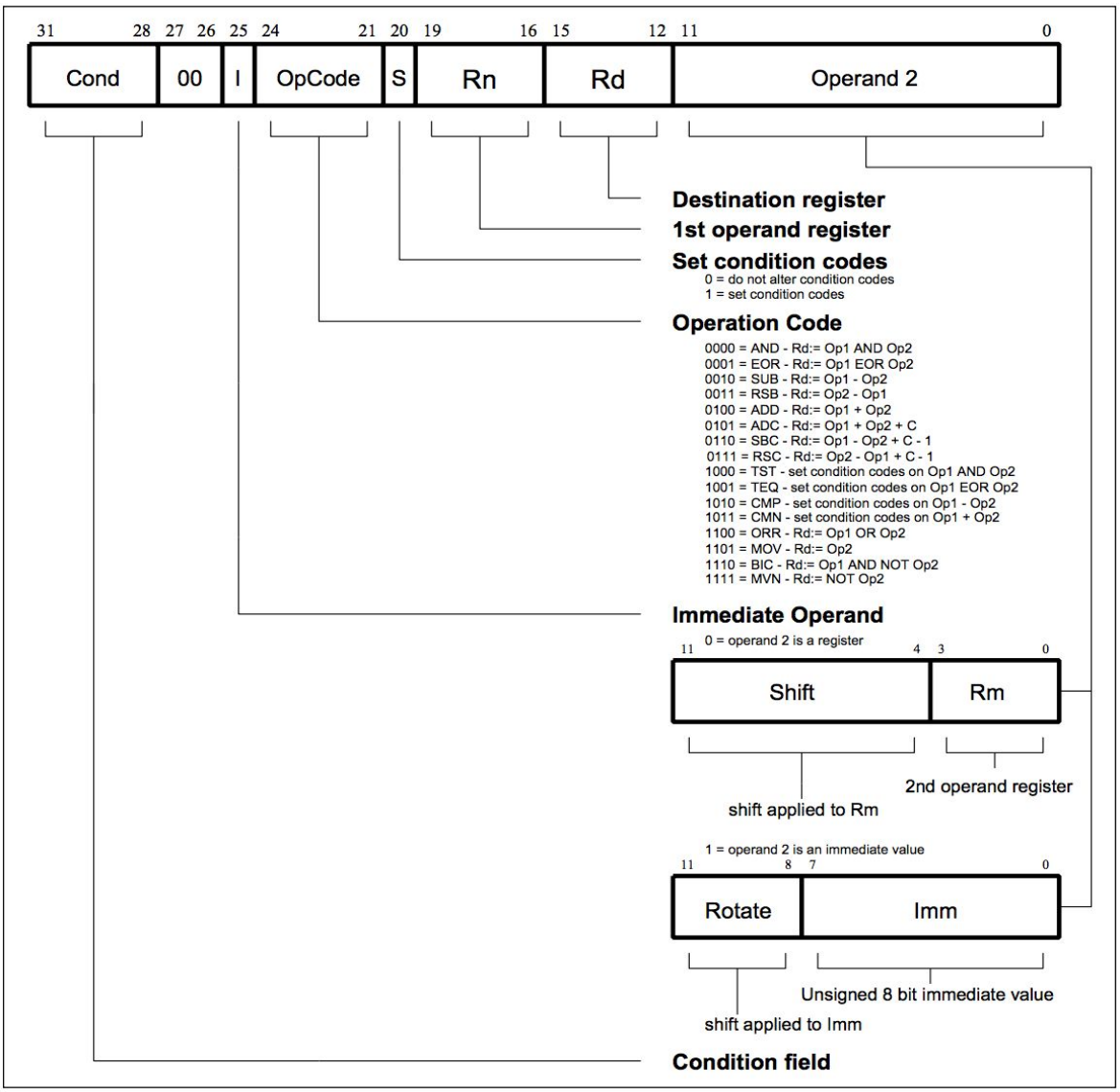
For a given instruction the 5 operations are executed sequentially, however different operations of different instructions can run parallely for faster overall execution. Let's have a look at these 5 phases in more detail:

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

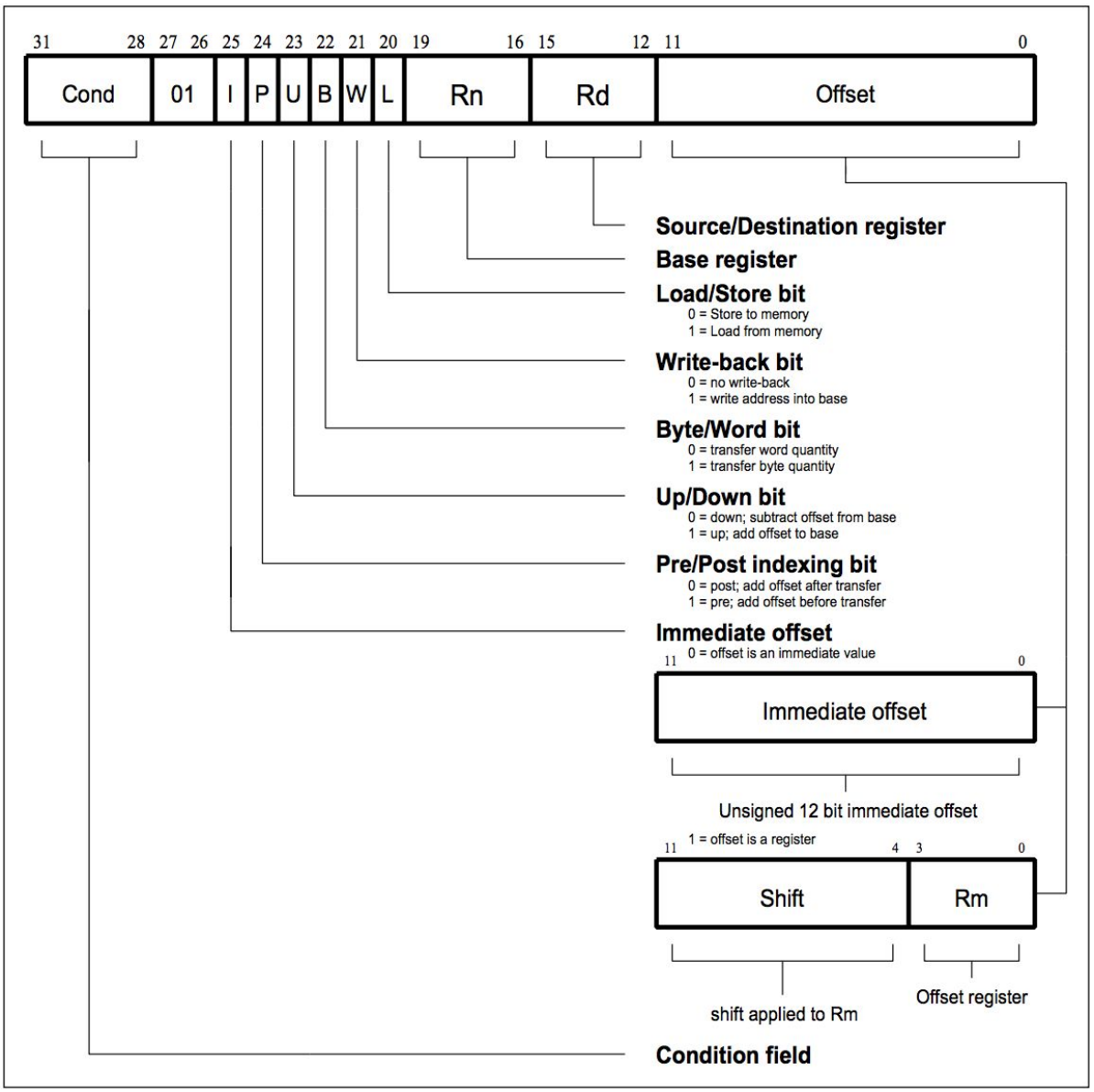
ARM instruction formats are as follows:
The first 4 bits (31,30,29,28) are for the **condition.**
Bits 27,26 indicate the **format** of instruction.

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.



Opcodes: ADD- 0100, SUB- 0010, MUL- 000A, MOV- 1101, CMP- 101

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.

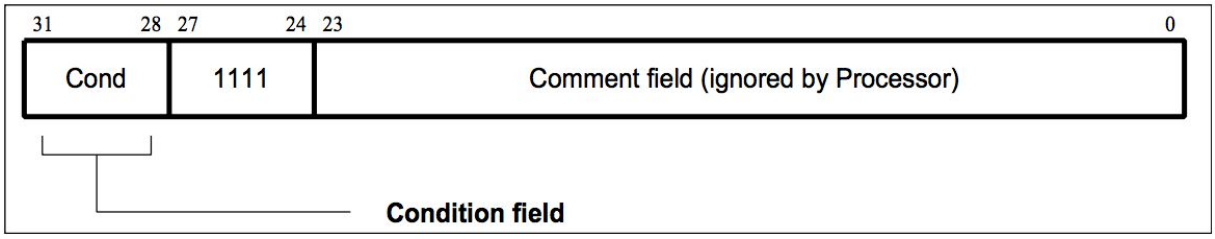


Ref: http://bear.ces.cwru.edu/eecs_382/ARM7-TDMI-manual-pt2.pdf

In both of the above cases, bits 19,18,17,16 denote Base register, and bits 15,14,13,12 denote Source/Destination register.

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.

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



- **Implementation Plan**

The steps are as follows:

- ➔ Read the input from the provided .MEM file line by line
- ➔ Parse the string to separate address and instruction
- ➔ Convert address and instruction to binary
- ➔ Create a mapping between address and instruction
- ➔ Create an array of registers
- ➔ Start execution of instruction by **fetching** the first instruction
- ➔ **Decode** the instruction by reading the bits 27, 26 to decide the instruction format
  - ◆ If they are 00, it denotes a data processing instruction which include arithmetic and logical instructions such as ADD, SUB, MUL, CMP, MOV etc. Further by reading the OPCode (bits 24-21), get the corresponding command to be executed. This is 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 are 01, it denotes a load/store instruction such as LDR, STR. Further by reading the 20th bit, conclude whether it is a load instruction or store instruction. Here, a memory location is obtained by reading the value in base register and getting the offset value. In case of a load, load the data

into the destination register from this memory location and in case of store, the value in the source register is stored into this memory location.

- If they are 10, it denotes a branch instruction. Here, the 24th bit indicates whether it is a branch, or a branch with link instruction. After this, get the location to branch to by reading the bits 23-0.

➔ **Execute** the data processing commands by performing the function of the arithmetic or logical instruction and calculate the results, storing them in temporary variables.

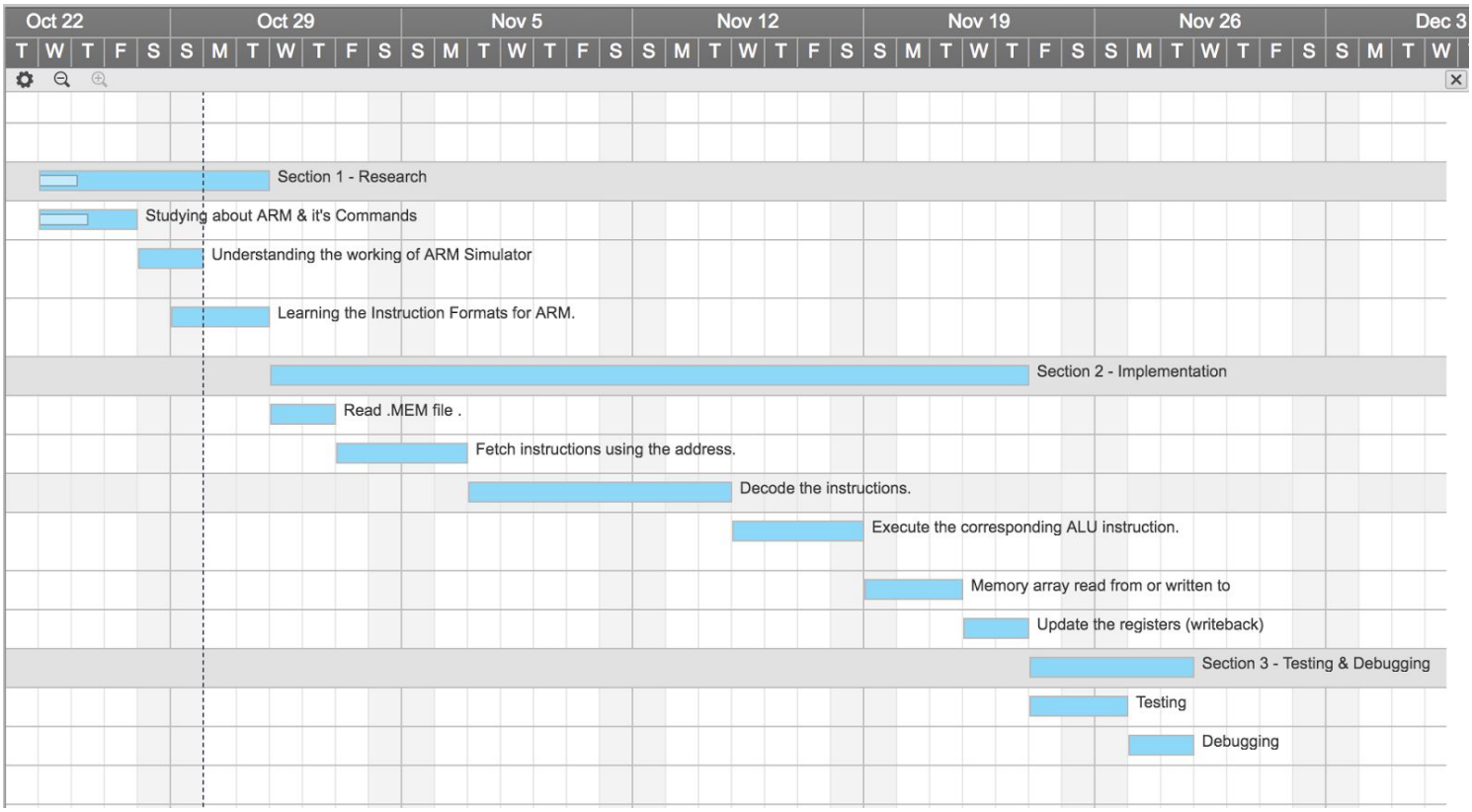
➔ **Memory** Load, Store instructions need to access the memory. Load/store data from or into the memory array.

➔ **Writeback** Based on the results of execute and load, accordingly update the array of registers and print the updated register values.

➔ Repeat the above steps for the next instruction

➔ Continue till "0xEF000011"

TIMELINE



----x----