Design Document: Functional Simulator for Subset of ARM instruction set

The document describes the design aspect of myARMSim, a functional simulator for subset of ARM instruction set.

# Input/Output

## Input

Input to the simulator is MEM file that contains the encoded instruction and the corresponding address at which instruction is supposed to be stored, separated by space. For example:

0x0 0xE3A0200A

0x4 0xE3A03002

0x8 0xE0821003

## Functional Behavior and output

The simulator reads the instruction from instruction memory, decodes the instruction, read the register, execute the operation, and write back to the register file. The instruction set supported is same as given in the lecture notes.

The execution of instruction continues till it reaches instruction “swi 0x11”. In other words as soon as instruction reads “0xEF000011”, simulator stops and writes the updated memory contents on to a memory text file.

The simulator also prints messages for each stage, for example for the third instruction above following messages are printed.

* Fetch prints:
  + “FETCH:Fetch instruction 0xE3A0200A from address 0x0”
* Decode
  + “DECODE: Operation is ADD, first operand R2, Second operand R3, destination register R1”
* Execute
  + “EXECUTE: ADD 10 and 2”
* Memory
  + “MEMORY:No memory operation”
* Writeback
  + “WRITEBACK: write 12 to R1”

# Design of Simulator

## Data structure

Registers, memories, intermediate output for each stage of instruction execution are declared as global static. Being static, the variables are not visible outside the file, thus, make the data encapsulated in the myARMSim.cpp.

## Simulator flow:

There are two steps:

1. First memory is loaded with input memory file.
2. Simulator executes instruction one by one.

For the second step, there is infinite loop, which simulates all the instruction till the instruction sequence reads “SWI 0x11”.

Next we describe the implementation of fetch, decode, execute, memory, and write-back function.

**FETCH:**

The FETCH function fetches commands from memory stored at the address given by program counter and stores it in **instruction\_word**. If the address is out of valid memory range ( 0x0 to 0xF9F ) or unaligned, the function halts the program.

**DECODE:**

The DECODE function scans **instruction\_word** for the functionality of the instruction. First it fetches the value from the register of the first operand and stores it in **operand1**. Then it stores the second operand (either from the register or immediate operand) to **operand2**. Then it stores the index of the result/destination/source register to **result\_reg**. Then it stores the value of status flag bit to **update\_flags**. The opcode and format of the instruction are stored in **Opcode** and **format** respectively. If format is BRANCH, it stores the 24-bit operand to **operand2**. If the instruction is comparison related (CMP,CMN,TST,TEQ), **update\_flags** is set to 1. For DT instruction, the immediate operand is inverted and **operand2** is recalculated (negative if required).

**EXECUTE:**

The EXECUTE command executes instructions of type DP,BRANCH and SWI. First, we check if the condition of the instruction is satisfied. For DP, we call the ALU with the operands and the 4-bit opcode which returns the answer and updates CPSR flags if **update\_flags** is set. We store the value in accumulator. For SWI, we check if the code is for exiting and then exit the program. For BRANCH, we change the PC value to PC + offset after sign\_extending and shifting twice.

**MEMORY:**

The MEMORY function executes instructions of type DT. First, we check if the condition of the instruction is satisfied. Then we check is **operand1+operand2** is a valid address (not out of range or unaligned). Then by checking the S bit, we do the desired memory operation. If LDR, we load the data from memory to accumulator. If STR, we store the data from the source register R{**result\_reg**} to the memory.

**WRITEBACK:**

The WRITEBACK function writes the data from accumulator to R{**result\_reg**} except on the following scenarios:

* If instruction is a comparison related (CMP,CMN,TST,TEQ).
* If the instruction is a STR instruction.
* If the instruction format is BRANCH or SWI.
* If the instruction condition is unsatisfied.

**Specifications and Miscellaneous Data**

* Executable File:./bin/myARMSim
* Registers: 16
* Memory Size: 32 KB (Memory address starting at 0x0).
* CPSR Registers: N,C,V,Z
* Stack Pointer : Memory Address 0xF9C
* Status flag support: Yes
* Instructions supported: AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN, STR, LDR, B, BL, SWI (Only 0x11).
* Conditions supported: EQ,NE,CS,CC,MI,PL,VS,VC,HI,LS,GE,LT,GT,LE,AL.
* Memory dump file: ./data\_out.mem
* Assembler log: STDOUT

# Test plan

We test the simulator with following assembly programs:

* Simple Adding program
* Fibonacci Program
* Sum of the array of N elements. Initialize an array in first loop with each element equal to its index. In second loop find the sum of this array, and store the result at Arr[N].
* Procedure Example: A program that uses BL to branch to a procedure and then revert back to the caller instruction.
* Fibonacci (Tail-recursion)