

Design Document: Functional Simulator for Subset of ARM instruction set

Developer's Name: Varun Jain, Varun Bansal
Developer's Email id: varun14170@iiitd.ac.in, varun13168@iiitd.ac.in
Date: 27th March 2015

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: Data Processing: OpCode is ADD, first operand R2, Second operand R3, destination register R1
 - Values R2 = 10, R3 = 2

- 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 (MEM) 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.

1. Load Program Memory
It fetches the instructions from the given .mem file and stores pointers to instructions in an array 'MEM'.
2. Fetch
By the use of function read_word we access the respective instructions. Instruction word is printed followed by address of the instruction which is stored in R[15]
3. Decode
After identification, opCode, operand and data registers are stored in variables.
Now, the instruction is identified. It either is a branch or data transfer or a data processing instruction.
 - a. For a Branch instruction, suitable type is identified (BEQ/ BNE/ BLE)
 - b. For a Data Transfer instruction, it is found whether the instruction is a Load or a Store instruction. Suitable offsets are calculated.
 - c. For a Data Processing instruction, it is found out if the instruction is immediate or a non-immediate addressed. In both cases, operand1 and operand2 are enumerated and suitable operand is printed (AND/ OR/ MOV)

4. Execute

- a. For a Branch instruction, the Condition Bits are checked to ascertain if we have to do a branch operation, and if so, the Program Counter is suitably incremented. Also, if the sign bit is '1', meaning a negative offset, the 2s complement of the offset is taken and then added to the program counter.

$$\text{New PC} = \text{Old PC} + (\text{adjusted offset}) * 4 + 4$$

- b. For a Data Transfer instruction, variable 'storage' is calculated which gets stored in write back.
- c. For a Data Processing instruction, 'storage' is calculated after suitable operations. In case of Compare, flags are updated.

The program counter is incremented for all instructions but the cases where a branch occurs.

5. Memory

For a Data Transfer instruction, the main memory MEM is updated with the correct register value in case of a Store instruction and the register is updated with a value from the MEM for a Load instruction.

6. Write Back

For all instructions other than Branch, the new calculated values are written back into the registers (register file).

Auxiliary Functions

7. Reset Proc

At exit, clears all the memory and register file.

8. Write Data Memory

Writes all the state of memory into a _out.mem file on exit.

9. ReadForRidonculous

Is a function that reads the specified bits from the instruction register.

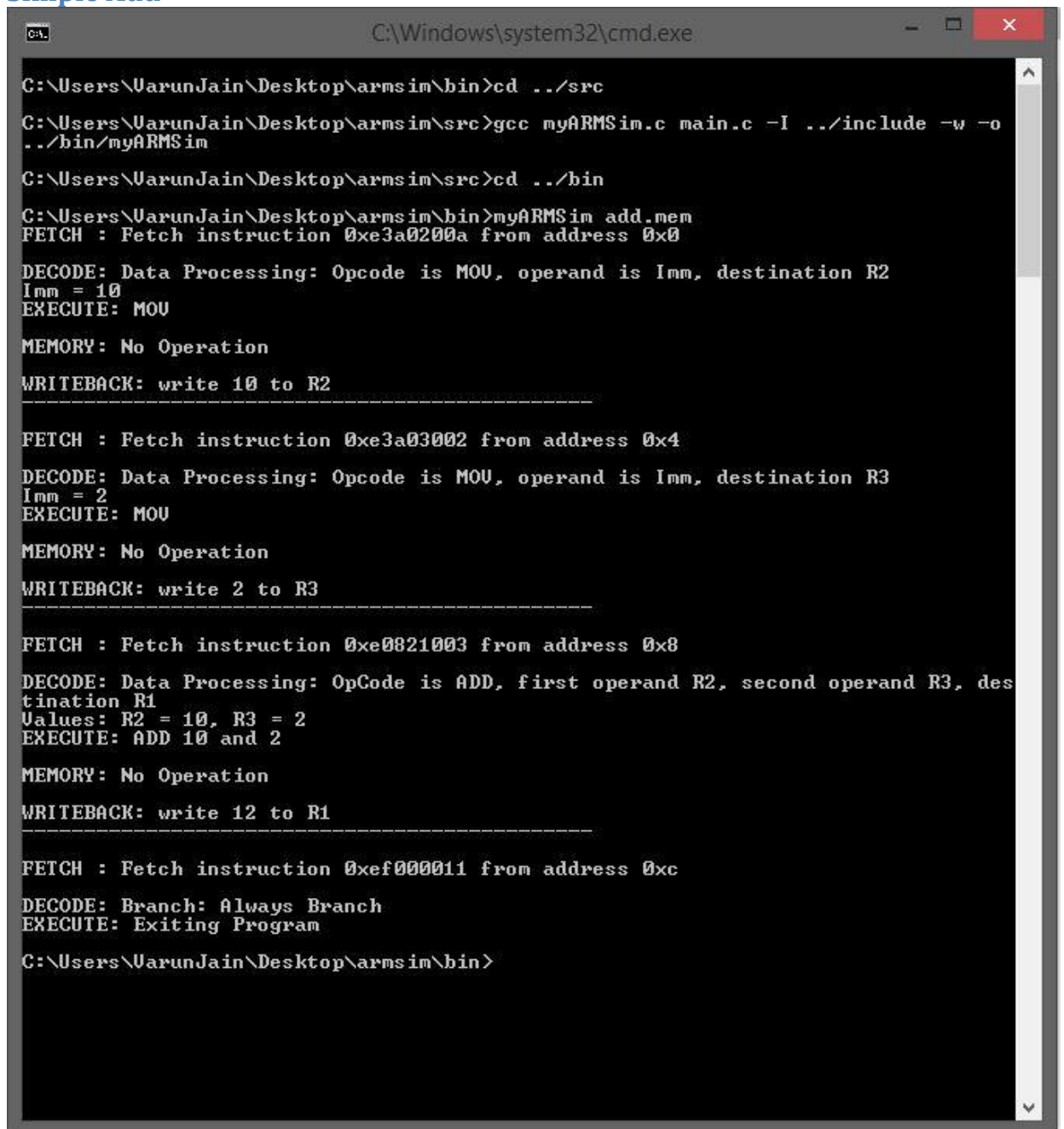
Test plan

We test the simulator with following assembly programs:

- Fibonacci Program: To print the first N Fibonacci numbers (1, 1, N-2)
- 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].

Screenshots for Some Test Plans

1. Simple Add



```
C:\Windows\system32\cmd.exe

C:\Users\VarunJain\Desktop\armsim\bin>cd ../src
C:\Users\VarunJain\Desktop\armsim\src>gcc myARMSim.c main.c -I ../include -w -o
../bin/myARMSim
C:\Users\VarunJain\Desktop\armsim\src>cd ../bin
C:\Users\VarunJain\Desktop\armsim\bin>myARMSim add.mem
FETCH : Fetch instruction 0xe3a0200a from address 0x0
DECODE: Data Processing: Opcode is MOV, operand is Imm, destination R2
Imm = 10
EXECUTE: MOV
MEMORY: No Operation
WRITEBACK: write 10 to R2
-----
FETCH : Fetch instruction 0xe3a03002 from address 0x4
DECODE: Data Processing: Opcode is MOV, operand is Imm, destination R3
Imm = 2
EXECUTE: MOV
MEMORY: No Operation
WRITEBACK: write 2 to R3
-----
FETCH : Fetch instruction 0xe0821003 from address 0x8
DECODE: Data Processing: OpCode is ADD, first operand R2, second operand R3, des
tination R1
Values: R2 = 10, R3 = 2
EXECUTE: ADD 10 and 2
MEMORY: No Operation
WRITEBACK: write 12 to R1
-----
FETCH : Fetch instruction 0xef000011 from address 0xc
DECODE: Branch: Always Branch
EXECUTE: Exiting Program
C:\Users\VarunJain\Desktop\armsim\bin>
```

2. Fibonacci (N = 6)

```
WRITEBACK: write 8 to R2
-----
FETCH : Fetch instruction 0xe1a01003 from address 0x38
DECODE: Data Processing: OpCode is MOV, operand R3, destination R1
Values: R3 = 13
EXECUTE: MOV
MEMORY: No Operation
WRITEBACK: write 13 to R1
-----
FETCH : Fetch instruction 0xeafffff5 from address 0x3c
DECODE: Branch: Always Branch
EXECUTE: Branch
MEMORY: No Operation
WRITEBACK: No operation
-----
FETCH : Fetch instruction 0xe3a04001 from address 0x10
DECODE: Data Processing: Opcode is MOV, operand is Imm, destination R4
Imm = 1
EXECUTE: MOV
MEMORY: No Operation
WRITEBACK: write 1 to R4
-----
FETCH : Fetch instruction 0xe3a05000 from address 0x14
DECODE: Data Processing: Opcode is MOV, operand is Imm, destination R5
Imm = 0
EXECUTE: MOV
MEMORY: No Operation
WRITEBACK: write 0 to R5
-----
FETCH : Fetch instruction 0xe3a06006 from address 0x18
DECODE: Data Processing: Opcode is MOV, operand is Imm, destination R6
Imm = 6
EXECUTE: MOV
MEMORY: No Operation
WRITEBACK: write 6 to R6
-----
FETCH : Fetch instruction 0xe1a01002 from address 0x1c
```

DECODE: Data Processing: Opcode is MOV, operand is Imm, destination R5
Imm = 0
EXECUTE: MOV

MEMORY: No Operation

WRITEBACK: write 0 to R5

FETCH : Fetch instruction 0xe3a06006 from address 0x18

DECODE: Data Processing: Opcode is MOV, operand is Imm, destination R6
Imm = 6
EXECUTE: MOV

MEMORY: No Operation

WRITEBACK: write 6 to R6

FETCH : Fetch instruction 0xe1a01002 from address 0x1c

DECODE: Data Processing: OpCode is MOV, operand R2, destination R1
Values: R2 = 8
EXECUTE: MOV

MEMORY: No Operation

WRITEBACK: write 8 to R1

FETCH : Fetch instruction 0xe1540006 from address 0x20

DECODE: Data Processing: OpCode is CMP, first operand R4, second operand R6
Values: R4 = 6, R6 = 6
EXECUTE: CMP 6 and 6. Also, Update Flags.
N= 0, Z= 1

MEMORY: No Operation

WRITEBACK: No writeback

FETCH : Fetch instruction 0xa0000007 from address 0x24

DECODE: Branch: ERROR
EXECUTE: MEMORY: No Operation

WRITEBACK: No operation

FETCH : Fetch instruction 0xef000011 from address 0x40

Exiting Program

C:\Users\VarunJain\Desktop\armsim\bin>