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”
  - “DECODE: Read registers 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.h.

- R[16] : 16 registers to process the instructions. Each register is a signed long integer 64 bits.
- MEM_INST[4000] : Main instruction memory where the the instructions are loaded. Each block is 8 bits (unsigned).
- MEM_HEAP[4000] : Heap memory for dynamic allocation. Each block is 8 bits (signed).

Simulator flow:

The simulator executes a program in 3 steps:

1. Memory is loaded with input memory file

   Files:

   init_memory.c
   load_program_memory.c

   Functions:

   - init_memory(): initializes all armsim variables to 0
   - load_program_memory(): reads the .mem input file and loads the instructions into instruction memory

2. Simulator executes instruction one by one.

   Simulator executes the instruction till it reaches end of Program Counter (PC) or if it encounters an 'swi 0x11' interrupt (0xEF000011).
   The simulator repeatedly calls fetch, decode, execute, memory and writeback while updating the PC and executes the instructions one by one.

3. Once the program finishes, the contents of the heap memory (MEM_HEAP) are written to an output memory file.

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

FETCH

Files:

fetch.c
readword.c

Work flow:

1. fetch() : fetches the current instruction word by calling read_word()
2. readword() : returns the current instruction word by fetching the instruction from memory corresponding to the current value of PC

DECODE

Files:

decode.c
decode_branch.c
decode_dataproc.c
decode_datatrans.c

Work flow:

1. decode() : extracts the bit specifying the type of instruction. Instruction type is as follows:

According to the instruction type, the respective function is called.

0	Data Processing
1	Data Transfer
2	Branch
3	SWI Exit

2. decode_dataproc() : called in case of a data processing instruction. This function extracts 5 values :

- Opcode
- Operand1
- Destination registers
- Immediate – specifies how operand 2 is given

If Immediate==1 : imm. reference

If Immediate==0 : register reference

- Operand2

3. decode_datatrans() : called in case of a data transfer instruction. Extracts whether the instruction is LOAD or STORE

4. decode_branch() : called in case of a branch instruction. Extracts the conditions bits along with the Zero and Negative flag to get what kind of branch it is.

Cond. Bits	Flags (N-negative Z-Zero)	Type of Branch
0	Z	BEQ
1	!Z	BNE
10	!N	BGE
11	N	BLT
12	!N && !Z	BGT
13	N || Z	BLE
14		B unconditional

EXECUTE

Files:

execute.c
execute_data_proc.c
execute_data_trans.c
execute_branch.c
update_flags.c

Work flow:

1. execute() : Executes 2 tasks :
    1. If instruction is 'swi 0x11' : Exit program
    2. Else, calls respective functions according to instruction type

2. execute_data_proc() : executes the operation based on the Opcode extracted during decode

Opcode	Operation (as in ARM instruction set)
0	AND

1	EOR
2	SUB
4	ADD
5	ADD with CARRY
10	SUBS (CMP)
12	ORR
13	MOV
15	MVN

3. execute_data_tans() : Extracts the base address ie- address of value to be stored in case of STR and address of destination register in case of LDR. Then updates the respective address

4. execute_branch() : Extracts the offset (signed) by which to change the PC for the current branch and applies the offset to PC.

MEMORY

Files:

mem.c
write_word.c

Work flow:
1. mem() : this is called only in case of data transfer instructions. It extracts the operand register or the memory offset. In case of an LDR instruction, the required value is loaded into the destination register (register parameters calculated by the the operand register and the offset).

2. write_word() : called in case of an STR instruction. Stores value in an intermediate variable to be written to memory in write back stage.

WRITEBACK

Files:

write_back.c

Work flow:

1. write_back(): If the instruction type is data processing, it writes back the answer to the destination register.

Test plan

Testing of the simulator is done using two sample programs given in the 'test' directory.

- Fibonacci Program: Finding the first 20 Fibonacci numbers using the iterative approach.
- 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].