Assignment-3 : Processor Simulator

Deadline : 24-Sept-2017

CS4110: Computer System Design Lab

1 Objective

To design a processor simulator.

2 Processor Configuration

Our processor has a 1KB memory. Assume that the memory is perfect, meaning that there won't be any memory misses. First half of the memory is used to store code and the remaining portion is used to store data. Assume that the processor has a register file (RF) with sixteen, 16-bit registers, named R0, ..., R15. Note that R0 always stores the value "0". The register file has 2 read ports and 1 write port. We consider a Reduced Instruction Set Computer (RISC) processor. The RISC architecture has the following instruction set.

- Three arithmetic instructions
  –
Instruction	**Effect**
ADD R1, R2, R3	R1 = R2 + R3
SUB R1, R2, R3	R1 = R2 - R3 done using two's complement arithmetic
MUL R1, R2, R3	R1 = R2 * R3
ADD R1, R0, #5	makes R1 = 5

  – Exactly one of the source operands of the arithmetic instruction can be a signed immediate operand of 4 bits stored in 2's complement format.

  – Here, we consider immediate addressing mode as well as register addressing mode.

- Two data transfer instructions
  –
Instruction	**Effect**
LD R1, A[Reg]	R1= content of the memory location (Reg+A)
SD A[Reg], R1	[Reg+A] = R1

  – Here, we consider displacement addressing mode.

- Two control transfer instructions

Instruction	Effect
JMP L1	Unconditional jump to location L1
BEQZ (Reg), L1	Jump to L1 if Reg content is zero

- L1 is given as an offset from current Program Counter (PC). This is called PC-relative addressing. L1 can be an 8-bit number represented in 2's complement format.

• Halt instruction

Instruction	Effect
HLT	Halts the computation

Maximum length of an instruction is 16 bits. We consider fixed length encoding for instructions. Note that at any point of time, the processor processes only one instruction. The address of the next in- struction to process is stored in Program Counter (PC), which is always incremented by 2 (as our ISA supports byte addressing). Any instruction fetched from memory is stored in Instruction Register (IR). The processor has to decode the instruction that is stored in IR and then execute the instruction.

3 Input and Output

Given an assembly code, the processor has to first load the corresponding binary code into memory (starting at address 0), followed by execution of each instruction to compute the task specified by the assembly code, and finally generate the correct output.

4 FAQs

1. Should 16 bit instructions be stored as short int? Or can we store it as an array of 16 chars?

   (a) Go for efficient implementation

2. How should the output look like?

   (a) See sample output file.

3. Which language to use ?

   (a) C/C++/Java

4. Should we simulate arithematic operations ? Or can we use the ones used by the high level language ?

   (a) No need to simulate processor level operations.

5. Input format ?

   (a) See sample input.
   (b) Read from stdin

6. Output Format ?
   (a) print to the stdout so that we can redirect.
   (b) Format of each line is as follows:
      i. <4-digit-decimal-address> <space> <colon> <space> <4-bits-of-data> <space> <last-4-bits-of-data>

7. Tips?
   (a) Choose appropriate data structures for memory and register file. For example, vectors or arrays in c++. Initialize them as per values given in input. Convert the given assembly code into binary format and store in memory using Big Endian Notation. Now read each instruction, **stored in binary format, from memory**, and depending on it's opcode, perform arithmatic operation on it's operands, using your programming language operators. At the end of program, print contents of your whole 1KB memory in the output file. **Make one submission per team.** If both the teammates submit, the evaluation script may fail. And you may incure penality.