Petri Net Simulator for a Simplified MIPS Processor

The Petri Net model will use colored tokens (token with values) rather than the default Petri net. The simulator generates step-by-step simulation of the Petri net model of the MIPS processor described below.

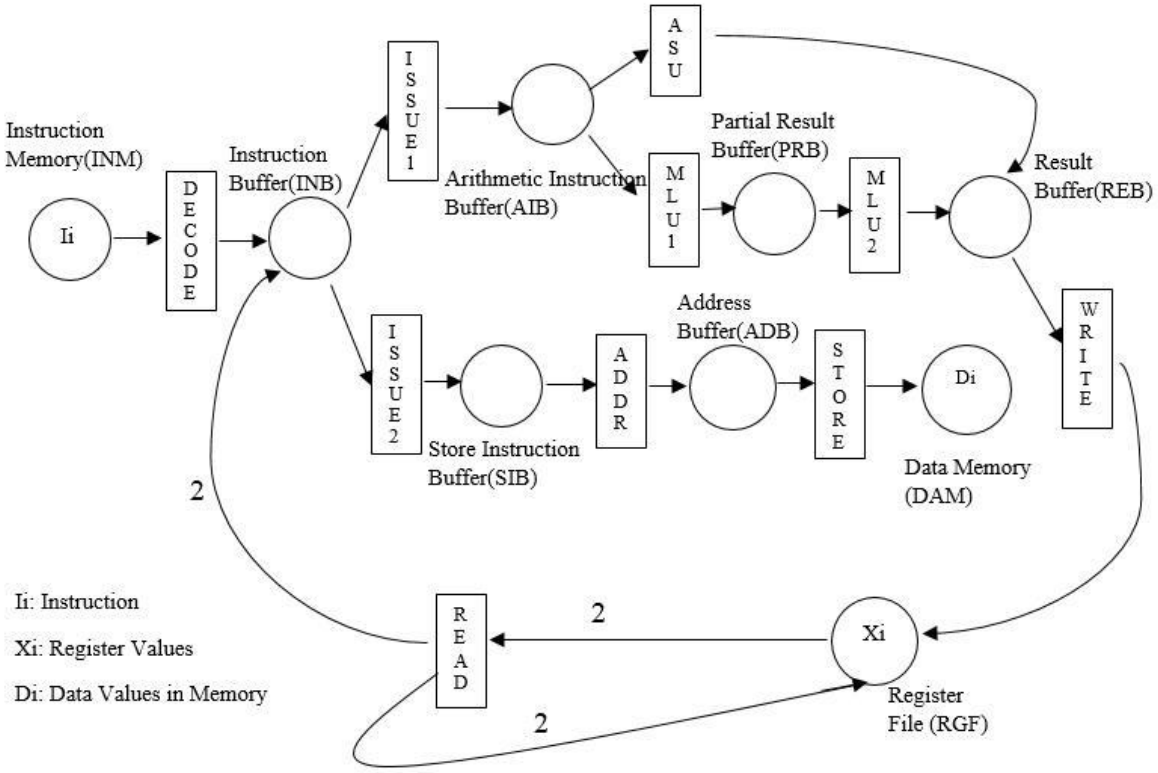

Figure 1. Petri Net Model of a MIPS

We first describe **three important places** (instruction memory, register file, and data memory) of the Petri net model. Next, we describe **ten transitions**. The remaining places can be viewed as buffers. All the arc carries (consumes) 1 token unless marked otherwise.

THREE IMPORTANT PLACES

1. Instruction Memory (INM):

The processor to be simulated only supports four types of instructions: Add (ADD), Subtract (SUB), Multiply (MUL) and Store (ST). At a time-step, the place denoted as Instruction Memory (INM) can have up to 16 instruction tokens.

<Opcode>, <Destination Register>, <First Source Operand>, <Second Source Operand>

Sample instruction tokens and equivalent functionality are shown below:

<ADD, R1, R2, R3> → R1 = R2 + R3
<SUB, R4, R3, R5> → R4 = R3 - R5
<MUL, R7, R2, R3> → R7 = R2 * R3
<ST, R7, R1, 4> → DataMemory [R1+4] = R7

2. Register File (RGF):

This MIPS processor supports up to 16 registers (R0 through R15). At a time step, it can have up to 16 tokens. The token format is <registername, registervalue>, e.g., <R1, 5>.

3. Data Memory (DAM):

This MIPS processor supports up to 16 locations (address 0 to 15) in the data memory. At a time step, it can have up to 16 tokens. The token format is <address, value>, e.g., <6, 5> implies that memory address 6 has value 5.

TEN TRANSITIONS

1. READ:

The READ transition is a slight deviation from Petri net semantics since it does not have any direct access to instruction tokens. Assume that it knows the top (in-order) instruction in the Instruction Memory (INM). It checks for the availability of the source operands in the Register File (RGF) for the top instruction token and passes them to Instruction Buffer (INB) by replacing the source operands with the respective values. For example, if the top instruction token in INM is <ADD, R1, R2, R3> and there are two tokens in RGF as <R2, 5> and <R3, 7>, then the instruction token in INB would be <ADD, R1, 5, 7> once both READ and DECODE transitions are activated. Therefore, both READ and DECODE transitions are executed together. Please note that when READ consumes two register tokens, it also returns them to RGF in the same time step (no change in RGF due to READ).

2. DECODE:

The DECODE transition consumes the top (in-order) instruction token from INM and updates the values of the source registers with the values from RGF (with the help of READ transition, as described above), and places the modified instruction token in INB.

3. ISSUE1:

ISSUE1 transition consumes one (in-order) arithmetic (ADD, SUB or MUL) instruction token (if any) from INB and places it in the Arithmetic Instruction Buffer (AIB). The token format for AIB is same as the token format in INB i.e., <opcode, dest register, source value1, source value2>.

4. ISSUE2:

ISSUE2 transition consumes one (in-order) store (ST) instruction token (if any) from INB and places it in the Store Instruction Buffer (SIB). The token format for SIB is same as the token format in INB i.e., <opcode, dest register, source value1, source value2>.

5. Add - Subtract Unit (ASU)

ASU transition performs arithmetic computations (ADD or SUB) and consumes one instruction (ADD or SUB) token (if any) from AIB, and places the result in the result buffer (REB). The format of the token in result buffer is same as a token in RGF i.e., <destination-register-name, value>.

6. Multiply Unit – Stage 1 (MLU1)

MLU1 consumes one instruction (MLU) token (if any) from AIB, and places the partial result in the partial result buffer (PRB). The token format for PRB is same as the token format in AIB i.e., <opcode, dest register, source value1, source value2>.

7. Multiply Unit – Stage 2 (MLU2)

Computes as per the instruction token from the PRB, and places the final result in the result buffer (REB). The format of the token in result buffer is same as a token in RGF i.e., <destination-register-name, value>.

8. Address Calculation (ADDR)

ADDR transition performs effective address calculation for the store instruction by adding the offset with the source operand. It produces a token as <register-name, data memory address> and places it in the address buffer (ADB).

9. STORE:

The STORE transition consumes a token from ADB and write the data to the data memory for the corresponding address. Assume that you will always have the data from the RGF. It places the data value in the Data Memory (DAM). The format of the token in DAM is already specified, i.e., <address, value>.

10. WRITE

Transfers the result (one token) from the Result Buffer (REB) to the register file (RGF). If there are more than one token in REB in a time step, the WRITE transition writes the token that belongs to the in-order first instruction.

Command Line and Input/Output Formats:

Input and output files as follows:

Instructions (input): instructions.txt
Registers (input): registers.txt
Data Memory (input): datamemory.txt
Simulation (output): simulation.txt

File Formats:

Inputs are provided in the specific format as listed below:

Input Register File Format: (see registers.txt for example)

```
<register name, value>
…
```

Input Data Memory File Format: (see datamemory.txt for example)

```
<memory address,data value>
…
```

Input Instruction Memory File Format (see instructions.txt for example):

```
<opcode,dest,src1,src2>
…
```

Step-by-step Snapshot Output File Format (see simulation.txt for example): **Please note that the following comments are not part of the output format**.

```
STEP 0:
INM: I1,I2,I3,…        # Where Ii are comma separated instruction tokens.
INB:                   # Comma separated tokens with source values.
AIB:                   # Comma separated arithmetic instruction tokens
SIB:                   # Comma separated store instruction tokens
PRB:                   # Comma separated instruction tokens
ADB:                   # Comma separated address tokens
REB:                   # Comma separated result buffer tokens
RGF:RF1,RF2,…          # Comma Separated register file tokens.
DAM:D1,D2,…            # Comma Separated data memory tokens.
<blank_line>
STEP 1:
```