Computer Architecture (ECE586)

Winter 2015

PDP-8 ISA Simulator

Tejas Patwa

Hanu Karshala

Ajinkya Kawale

Akshay Shah

**Introduction:**

PDP-8 ISA Simulator implements all the instruction types with different addressing modes as stated in the problem statement. An ISA simulator is a useful tool to debug ASM and also generating statistics like Average CPI, Instruction counts, Clock cycles for different instruction types, trace file and branch trace for Performance Benchmarks.

**Features:**

1. MRI, I/O Instructions (except interrupt system) and MicroOps (except group 3) are supported.
2. Printing of Data memory accessed during execution is available.(valid bit implementation)
3. Trace and Branch trace generated as text files.
4. Instruction counts for all type of instructions decoded.
5. Accurate clock cycles simulated for ASMs.
6. Breakpoints can be included/deleted at program addresses through a text file.
7. Single stepping supported with printing machine status information.
8. Teletype printer and Keyboard simulated using input from text file.

**Assumptions:**

1. All the ASM Program’s start address is octal 0200.
2. ISZ instruction does not account for a read for testing {Value [EA] ==0}.
3. Illegal instruction combinations of instruction mnemonics by programmer may result in legal machine codes.
4. Not supported and illegal instructions are treated as NOPs.
5. Not supported, illegal instructions and extra credit instructions does not account for clock cycles. (Although keeping a count for each type).
6. A zero value in Accumulator is considered as a positive number.
7. Opcode 7 instructions combinations compiles to one instruction word and it accounts for one clock cycle.

**Implementation:**

Mem read /write functions:

Implemented tasks for memory read/write for every memory access. This helps for trace generation and breakpoints debugging.

Effective address:

Effective addresses were generated by a task using PC and offset respective to the addressing mode decoded.

Trace generation:

A task for trace generation called for every memory access and writes a trace in text file (trace.txt)

Branch Trace generation:

Branch trace task generates a text file with detailed information for branch instructions like taken, not taken conditional and unconditional.

Instructions counts and clock cycle:

Instructions as per the documentations are implemented for all instructions using tasks. These tasks were used to increment instruction counts and clock cycle counts depending on the addressing mode (incremented in effective address task).Counter array used for different types of instructions.

I/O:

I/O instructions for keyboard and teletype printer implemented using approximate simulation of keyboard and printer. These I/O Devices uses buffers and flags for communicating.

Debugging:

Breakpoints can be included or single stepping can be turned on for debugging purpose.

Memory printing:

Valid bit logic implemented for printing memory slots which are accessed in the execution.

**Testing:**

Instructions:

We tested instructions exception for most of the corner cases by running different asm files including MRI, I/O, MicroOps and MicrOps combinations.

Addressing modes:

We tested for different addressing modes using MRI instructions. Auto indexing was tested for block transfer. JMS instruction used to test indirect/direct current or zero page addressing modes.

Trace, Counters and clock cycles:

Trace file generated was tested for random test cases and branch trace tested visually on traces generated from branch instructions. Counters for all instruction type where tested for some test cases. Also accurate clock cycle count was tested for different opcodes and addressing modes.

Debugging:

Breakpoints and single step debugging tested on a random test case.

Keyboard printer:

Keyboard read and printer printing simulation using PDP 8 protocol tested using key inputs from text file and printing using $display.

Illegal/not supported instructions:

Illegal/not supported instruction were included in test cases to test the expected behaviour of the simulator.