Tutorial 4

EduMIPS64

Course Project

- Worth 25% of the overall grade
- Milestones
  - Project Proposal (10%) – due March 1
  - Progress Report (15%) – due March 17 (midnight)
  - Final Report (60%) – due April 12
  - Team-mate grades (15%) – due April 12
- Java Programming Involved
- Could take a while. Start Early!

EduMIPS64

- Open Source MIPS64 Simulator
- Written in Java
- Executes Assemble Programs
- Compiled using Apache ANT
- Basic I/O facilities
  - Read/Write to/from the console
  - Read/Write to/from files

Obtaining the software

- All available on Mycourses
- Extract the files
- Build using the readme file instructions
- Run the simulator
- Run sample assembly language codes provided to visualize the transition of instructions and data

Using EduMIPS64

- Read the manual (On Mycourses)
- Run sample assembly language codes provided to visualize the transition of instructions and data
- Will not run assembly programs generated by a cross compiler (eg. gcc)
- Displays
  - Timing Diagrams, pipeline state
  - Register file / memory contents
  - RAW hazards
  - CPI
- Allows you to step through the code and see the resulting changes on the processor state.

MIPS64 Assembly

- `; This is a comment`
- `.data`
- `label: .word 15 ;This is an inline comment`
- `.code ;or .text`
- `daddi r1, r0, 1`
- `syscall 0 ; TRAP 0; HALT`

MIPS64 Assembly

- Two Sections
  - Data - where you store the data on which you compute (corresponds to memory)

    It stores bytes (1B), half-words (2B), words (4B) or double-words

  - Code – It is where the program actually lives
    - 32 integer registers (R0-R31)
    - Operands can be
      - Registers
      - Immediate Values
      - Addresses

MIPS64 Instructions

- ALU instructions
  - AND, DADD, DADDU, DDIV, ...
- Load/store instructions
  - LD, SD, ...
- Flow control instructions
  - J, JR, B, BEQ, ...
- System calls
  - exit, open, close, read, printf, ...
- Other
  - BREAK, NOP, HALT

Code Structure

- Directories
  - **core/**: main code of the simulator
  - data/: some documentation
  - docs/: main documentation
  - libs/: external libraries used by the simulator
  - ui/: user interface
  - utils/: exception classes, translations, user settings
  - test/: validation test programs

Code Structure

- core/: main code of the simulator
- Parser.java: parse the assembly source file
- BitSet {32,64}.java: represent a 32/64-bit quantity
- CPU.java: top-level entity representing the processor
  - SymbolTable.java: map labels to data/instructions
  - Register.java: single GPR element (R0 special case)
  - Memory.java: data and instruction memories
- MemoryElement.java: one data memory element
- is/Instruction.java: one instruction
- IOManager.java: process open/read/... system calls

Code Structure

- core/is/: instructions definitions
  - Instruction.java: generic definition of an instruction
    - ALUInstructions.java: ALU

      ALU_{I,R}Type.java

      ADDU.java, ...
    - FlowControlInstructions.java: Flow control

      FlowControl_{I,J,R}Type.java

      BNE.java, ...
    - LDSTInstructions.java: Load/Store

      Loading.java, Storing.java

      LD.java, ...
- InstructionUtils.java: define binary ALU operations

Testing your changes

- The modifications you make to the simulator must not impact the outcome of any program
- You are expected to test your modifications thoroughly
  - Determine what might break
  - Design test programs to validate those cases
- You may also validate your changes against the unmodified version of the simulator
- See test/ for sample test programs