Mobile Processor Project #2: Single-cycle MIPS

Mobile Processor Programming assignment #2: Single-cycle MIPS

due by 4.17

Introduction

Now, we will build a MIPS simulator. MIPS is a microprocessor that swept last decades. It has simple ISA, and optimized for concurrent parallel execution for performance. MIPS ISA defines 31 integer instructions. MIPS instructions are categorized into I, R, J types, and MIPS ISA defines 32 general-purpose registers. Extending the concepts from simple calculator, we are building a CPU simulator that takes MIPS executable binary (input.bin) as input and emulates the operation in single-cycle MIPS. MIPS binary file can be generated from mips-cross compiler. You can find more details of MIPS ISA, in the MIPS green summary sheet.
The followings are requirements for your program.

1. To make MIPS binary, use MIPS cross compiler toolchain. First, write C code, and compile mips-linux-gnu-gcc with –c option (compile only). Second, translate the object binary, stripping ELF headers. mips-linux-gnu-objcopy –O binary –j .text input.o input.bin. Third, check the integrity between binary files: objectdump, mips-linux-gnu-objdump –d input.o , vi –b input.bin → check with :%!xxd option. (google internet) or hexedit tool
   A. To use with multiple functions, you need to hand-carve binary for function calls (jal 0, originally). For example, jal to 0x40 can be encoded as (0x0C000010).
2. Before the execution, the binary file is loaded into the memory. Note that memory can be a data structure defined with large array. Read all the file content into your memory (data structure). Assume that all register values are all zero, except for LR(r31) and SP(r29). Initial LR value is 0xFFFF:FFFF. Thus, when your PC becomes 0xFFFF:FFFF, your machine completes execution, and halts. Your application is loaded to 0x0, and initial stack pointer(r29) is 0x800000.
3. MIPS executes instructions in the following stages:
   A. Instruction fetch, at which stage, instruction is moved from memory to CPU,
   B. Instruction decode, at which stage, instruction is decoded,
   C. Execution, at which stage, ALU operates, and the calculation result is out,
   D. Load/store result to memory, at which stage, load/store memory operation is completed.
   E. Write back to reg. file, update register values. (including PC)
4. Single cycle MIPS processor performs all the above five stages in the same cycle. The simulator could implement functions that correspond to stages. To begin the machine, it moves PC value to the very beginning point of the program, which is the address zero.
5. At the end of each cycle, the simulator prints out the changed architectural state from the previous cycle. User visible architectural state consists of set of general-purpose registers, PC, and memory.
   A. You can print out only changed state.
6. At the end of the execution, you need to print out the calculated result value. We know the end of execution by moving PC to 0xFFFF:FFFF.
   A. The final return value is stored in v0 (or r2) register.
7. Sample programs are uploaded to the course web site. (E-learning site.)
   A. Simple return (simple.c)
   B. Return with value 100 (simple2.c)
   C. Return the sum from 1 to 100 (simple3.c) – up to here, basic requirement
   D. Return the sum from 1 to 100 using recursion (simple4.c)
   E. Calculate the GCD of 0x1298 and 0x9387 (gcd.c)
   F. Calculate 10th Fibonacci number (fib.c) – and more, you can make more calculation program
   G. Return 102-th smallest number from 10000 random numbers. (input4.c)
8. Output
   A. For each instruction execution,
      i. Changed architectural state
   B. After the completion of the program
      i. Final return value (value in r2) – up to here, basic requirement
      ii. Number of executed instructions
      iii. Number of (executed) R-type instruction

   - iv. Number of I-type instruction
   - v. Number of J-type instruction
   - vi. Number of memory access instructions
   - vii. Number of taken branches

9. Optionally, you can implement separated control signals from data path.
10. Optionally, you can put additional instruction JALR (jump and link using register)
    - A. Bit-encoding
      - i. Rd: 31 for saving return address
      - ii. Rs: jump target address
      - iii. Funct: 0x9
      - iv. Rest of bits are zero
    - B. Semantics
      - i. Saving return address to Rd, and
      - ii. Jump to Rs address
    - C. Sample program
      - i. You have to make sample program. Consider fib.bin's jal to jalr.
      - ii. You have to fix binary.
      - iii. Make sure that your program runs correctly.

Hope to enjoy MIPS emulator programming!