C++/Project 1 Overview

COS 375

Project 1: Overview

- Create a high level functional simulator for a subset of the MIPS ISA

- DON’T create individual components of the datapath (e.g. ALU, muxes)

- DO keep track of architectural state during execution

Project 1: Steps

- Read in a binary file and store to MemoryStore object
- Set $pc to 0 and start executing instructions
- Update register and memory states as needed by each instruction
- Halt when trying to execute instruction encoded as 0xfeedfeed
- Print memory and register states

```
main (argc, argv) {
  Create a memory store called myMem
  Initialize registers to have value 0
  Read bytes of binary file passed as parameter into appropriate memory locations
  Point the program counter to the first instruction
  while (TRUE) {
      Fetch current instruction from memory@PC
      Determine the instruction type
      Get the operands
      switch (instruction type) {
      case 0xfeedfeed:
          RegisterInfo reg;
          Fill reg with the current contents of the registers
          dumpRegisterState(reg);
          dumpMemoryState(myMem);
          return 0;
       case INSTR1:
          Perform operation and update destination
          register/memory/PC
          break;
       ...
      default:
          fprintf(stderr,"Illegal operation...");
          exit(127);
      }
   }
}
```

RegisterInfo

- Used for storing register state before calling dumpRegisterState()

NAME	NUMBER	USE
$zero	0	The Constant Value 0
$at	1	Assembler Temporary
$v0-$v1	2-3	Values for Function Results and Expression Evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved Temporaries
$t8-$t9	24-25	Temporaries
$k0-$k1	26-27	Reserved for OS Kernel
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

```
struct RegisterInfo
{
    //The $at register.
    uint32_t at;
    //The $v registers.
    uint32_t v[V_REG_SIZE];
    //The $a registers.
    uint32_t a[A_REG_SIZE];
    //The $t registers.
    uint32_t t[T_REG_SIZE];
    //The $s registers.
    uint32_t s[S_REG_SIZE];
    //The $k registers.
    uint32_t k[K_REG_SIZE];
    //The $gp register.
    uint32_t gp;
    //The $sp register.
    uint32_t sp;
    //The $fp register.
    uint32_t fp;
    //The $ra register.
    uint32_t ra;
};

extern void dumpRegisterState(RegisterInfo & reg);
```

RegisterInfo

- Used for storing register state before calling dumpRegisterState()
- Don’t use this struct as your registers while executing!

```
struct RegisterInfo
{
    //The $at register.
    uint32_t at;
    //The $v registers.
    uint32_t v[V_REG_SIZE];
    //The $a registers.
    uint32_t a[A_REG_SIZE];
    //The $t registers.
    uint32_t t[T_REG_SIZE];
    //The $s registers.
    uint32_t s[S_REG_SIZE];
    //The $k registers.
    uint32_t k[K_REG_SIZE];
    //The $gp register.
    uint32_t gp;
    //The $sp register.
    uint32_t sp;
    //The $fp register.
    uint32_t fp;
    //The $ra register.
    uint32_t ra;
};

extern void dumpRegisterState(RegisterInfo & reg);
```

MemoryStore

- Header file contains interface — implementation in UtilityFunctions.o

Choose right one

```
//The memory is 64 KB large.
#define MEMORY_SIZE 0x10000

//The various sizes at which you can manipulate the memory.
enum MemEntrySize
{
    BYTE_SIZE = 1,
    HALF_SIZE = 2,
    WORD_SIZE = 4
};

//A memory abstraction interface. Allows values to be set and retrieved at a number of
//different size granularities. The implementation is also capable of printing out memory
//values over a given address range.
class MemoryStore
{
    public:
        virtual int getMemValue(uint32_t address, uint32_t & value, MemEntrySize size) = 0;
        virtual int setMemValue(uint32_t address, uint32_t value, MemEntrySize size) = 0;
        virtual int printMemory(uint32_t startAddress, uint32_t endAddress) = 0;
        virtual ~MemoryStore() {}
};

//Creates a memory store.
extern MemoryStore *createMemoryStore();

//Dumps the section of memory relevant for the test.
extern void dumpMemoryState(MemoryStore *mem);
```

Pass by reference!

C struct vs C++ struct

C	C++
Can hold non-static data	Can hold member functions and static members
Need to initialize fields externally	Constructors built in
All fields are public	Can specify access modifiers
structs cannot easily reuse other structs	structs can inherit other structs
Need to use `struct` keyword when declaring variables	Just use the name of the struct

C++ tidbits

- C++ structs and classes are (almost) the same thing

- std::cout == printf() but more versatile and simpler syntax
  - `cout << "The 32-bit (word) value of address 0x10 is 0x" << hex << setfill('0') << setw(8) << value << endl;`

- Pass by reference allows you to get around having to pass pointers

- gdb is still your best friend for debugging 🙃

Tools

- mips-linux-gnu-as: assemble assembly code to machine code (ELF format)
  - `mips-linux-gnu-as fib.asm -o fib.elf`

- mips-linux-gnu-objcopy: copy sections of binary/ executable file
  - `mips-linux-gnu-objcopy fib.elf -j .text -O binary fib.bin`

- mips-linux-gnu-objdump: disassemble ELF file for inspection
  - `mips-linux-gnu-objdump -D fib.elf > fib.dump`

Questions?

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)