

## Operating Systems Design (CS 423)

Elsa L Gunter  
2112 SC, UIUC

<http://www.cs.illinois.edu/class/cs423/>

Based on slides by Roy Campbell, Sam King, and  
Andrew S Tanenbaum

3/28/11

1

## Goldberg '74

- A virtual machine is...
- "an efficient, isolated, duplicate of the real machine"
- VM environment created by VMM
- Guest == virtual
  - Guest OS and apps run INSIDE a VM
  - Guest OS runs ABOVE a VMM
- Host == physical

3/28/11

2

## VMM environment

- Duplicate
  - Virtual machine == physical machine
- Efficient
  - Runs almost as fast as real machine
- Isolated
  - VMM has control over resources
  - What are the resources the VMM controls?

3/28/11

3

## VMM control and isolation

- Guest software can only access resources allocated to it
- VMM can reclaim resources
- E.g., More guest physical mem than phys mem

3/28/11

4

## VMM research

- Record and replay
  - Debugging OSes
- Migration
  - Moka5 user convenience,
  - VMmotion for reliability
- Security
  - NSA NetTop
  - Isolated email and browser
  - Intrusion detection

3/28/11

5

## VMM as Simulators

- "A computer is a large state machine"
- Simulator state is easy to work with
- Instruction-by-instruction simulation is slow

3/28/11

6

## VMM is a simulator

- Simulator is software that simulates the execution of a computer

```
while(1){
    inst = mem[PC]; // fetch
    if(inst == add) { // decode// execute
        reg[inst.reg1] = reg[inst.reg2] + reg[inst.reg3];
        PC++;
    }
    else if ...
} // repeat
```

3/28/11

7

## Simulator

- Disk → file
- Memory → stack
- Display → window

3/28/11

8

## Simulation

- Lots of real instructions for single virtual instruction
- Many instructions more complicated
  - Update condition codes
  - Memory loads / stores
  - Exceptions
- Question: how can we make this fast?
- First, get a simulator ...

3/28/11

9

## Our Sparc Simulator

- 32 registers
- R0 is always 0
- RISC architecture
  - Load/store only instructions to access memory
  - Around 75 instructions total

3/28/11

10

## Our Sparc simulator

- Simplified, no register windows
- No npc, a few other registers missing
- Does use branch delay slots
- Assume underlying endianness is same as simulated endianness
  - Modified sparc binary to little endian

3/28/11

11

## Code from Sam King

```
typedef uint32_t u32;
typedef int32_t i32;
// Visible state of CPU
struct CPU {
    u32 regs[NUM_REGS];
    u32 pc;
    bool icc_z;
    bool icc_n;
};
```

3/28/11

12

## Code from Sam King

```
struct control_signals {
    u32 op;
    u32 rd;
    u32 cond;
    u32 op2;
    u32 op3;
    u32 rs1;
    u32 i;
    u32 asi;
    i32 simm;
    u32 imm;
    u32 disp22;
    u32 disp30;
    u32 rs2;
    u32 raw;
};
```

3/28/11

13

## Sparc Simulator – Sam King

```
int main(int argc, char *argv[]) {
    ifstream infile;
    struct CPU *cpu;
    ...
    signal(SIGINT, ctrlC);

    // initialize our state
    ram = new uint8_t[RAM_SIZE];
```

3/28/11

14

## Sparc Simulator – Sam King

```
cpu = new struct CPU;
for(int idx = 0; idx < NUM_REGS; idx++) {
    cpu->regs[idx] = 0;
}
cpu->pc = 0;
// setup the stack pointer
cpu->regs[14] = RAM_SIZE-4-120;
// setup the fib program
cpu->regs[8] = RAM_SIZE-4;
```

3/28/11

15

## Sparc Simulator – Sam King

```
// fetch our memory image and cpu state (if set)
fillState(argv[1], ram, RAM_SIZE);
if(argc >= 3) {
    fillState(argv[2], cpu, sizeof(struct CPU));
}

startTime = time(NULL);
cpu_exec(cpu); // Starts with fetch
return 0;
}
```

3/28/11

16

## Task 1: Implement fetch

```
u32 fetch(struct CPU /*cpu*/) {
    u32 opcode = 0;
    // XXX FILL THIS IN

    return opcode;
}
```

3/28/11

17

## Task 1: Implement fetch

```
u32 fetch(struct CPU /*cpu*/) {
    u32 opcode = 0;
    assert((cpu->pc & 0x3) == 0);
    assert(cpu->pc < RAM_SIZE);
    memcpy(&opcode, ram+cpu->pc, sizeof
(opcode));
    //ram pointer to 8 bit value; opcode 32 bit
    return opcode;
}
```

3/28/11

18

## Task 2: Implement exec

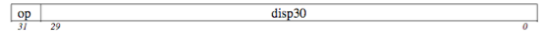
- Decode instructions
- Update register state
- Update pc state
- Note: r0 is always 0!
- Note: advancing the pc in nonbranch
  - pc+=4

3/28/11

19

## Sparc instruction format

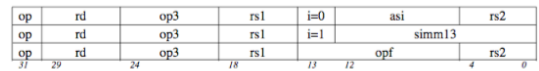
Format 1 (op = 1): CALL



Format 2 (op = 0): SETHI & Branches (Bicc, FBficc, CBccc)



Format 3 (op = 2 or 3): Remaining instructions



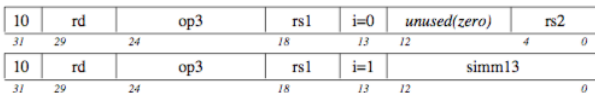
3/28/11

20

## Add format

| opcode | op3    | operation                     |
|--------|--------|-------------------------------|
| ADD    | 000000 | Add                           |
| ADDcc  | 010000 | Add and modify icc            |
| ADDX   | 001000 | Add with Carry                |
| ADDXcc | 011000 | Add with Carry and modify icc |

Format (3):



3/28/11

21

## Addi

- $r[rd] = r[rs1] + \text{sign\_ext}(\text{sim13})$

```
void addi(struct CPU *cpu, u32 rd, u32 rs1, i32
signedImm) {
    cpu->regs[rd] = cpu->regs[rs1] +
signedImm;
```

3/28/11

22

## Task 2: Implement exec

- Decode instructions
- Update register state
- Update pc state
- Note: r0 is always 0!
- Note: advancing the pc in nonbranch
  - pc+=4

3/28/11

23