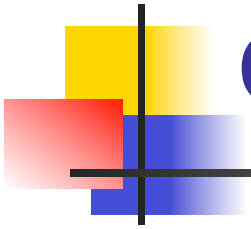
Operating Systems Design (CS 423)

Elsa L Gunter

2112 SC, UIUC

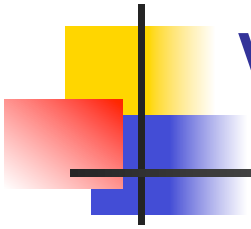
http://www.cs.illinois.edu/class/cs423/

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



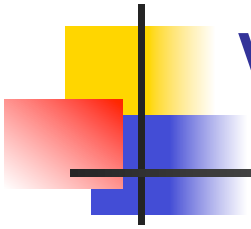
Goldberg '74

- 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



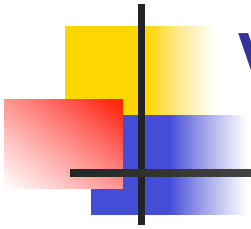
VMM environment

- 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?



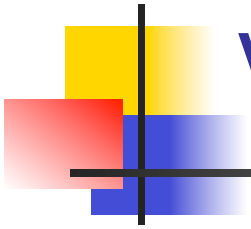
VMM control and isolation

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



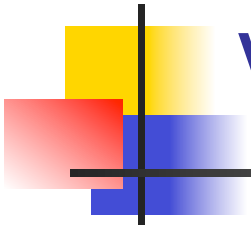
VMM research

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



VMM as Simulators

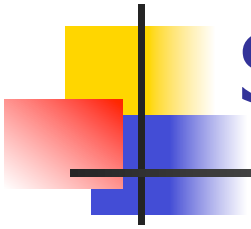
- "A computer is a large state machine"
- Simulator state is easy to work with
- Instruction-by-instruction simulation is slow



VMM is a simulator

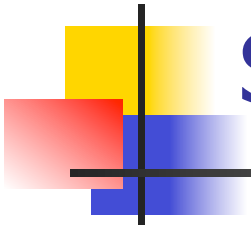
- Simulator is software that simulates the execution of a computer

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



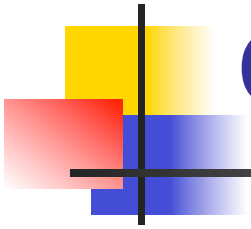
Simulator

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



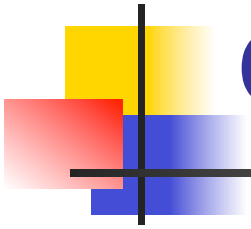
Simulation

- 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 …



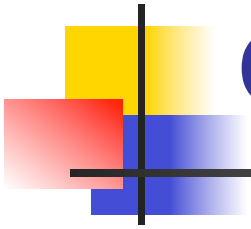
Our Sparc Simulator

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



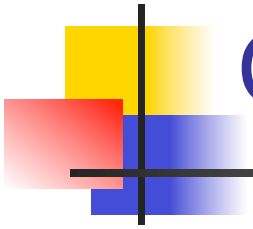
Our Sparc simulator

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



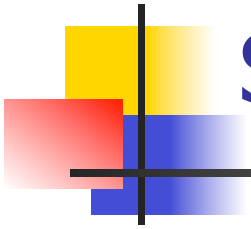
Code from Sam King

```
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;
};
```



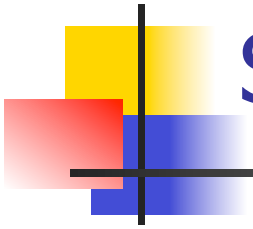
Code from Sam King

```
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;
};
```



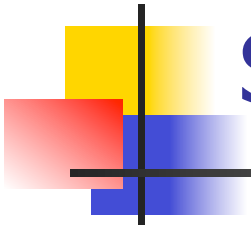
Sparc Simulator – Sam King

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

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



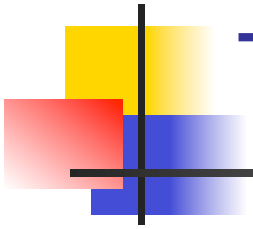
Sparc Simulator – Sam King

```
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;
```



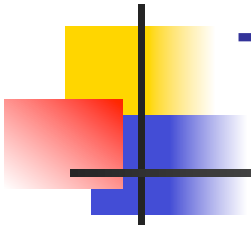
Sparc Simulator – Sam King

```
    // 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;
}
```



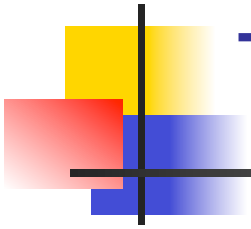
Task 1: Implement fetch

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


    return opcode;
}
```



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 valiue; opcode 32 bit
    return opcode;
}
```



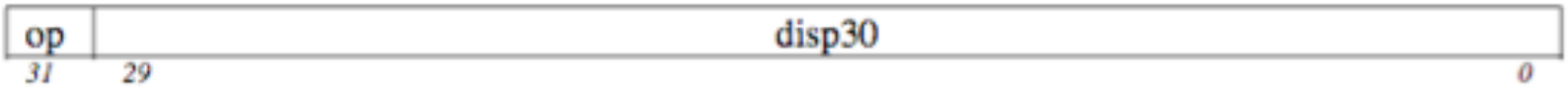
Task 2: Implement exec

- Decode instructions
- Update register state
- Update pc state

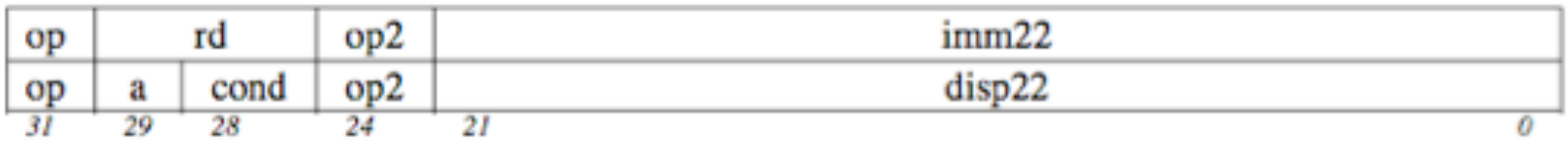
- Note: r0 is always 0!
- Note: advancing the pc in nonbranch
  - pc+=4

Sparc instruction format

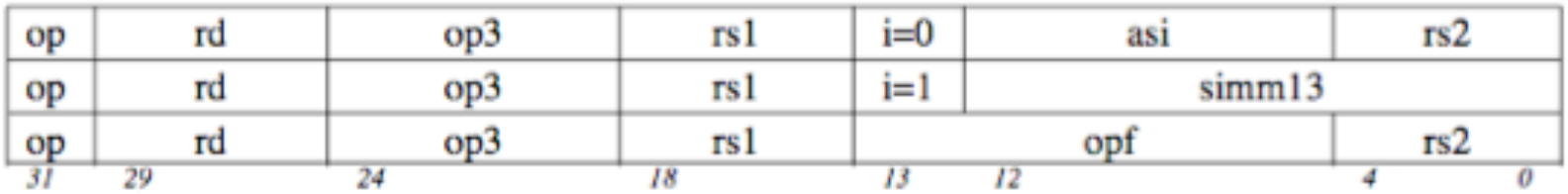
Format 1 (*op* = 1): CALL



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



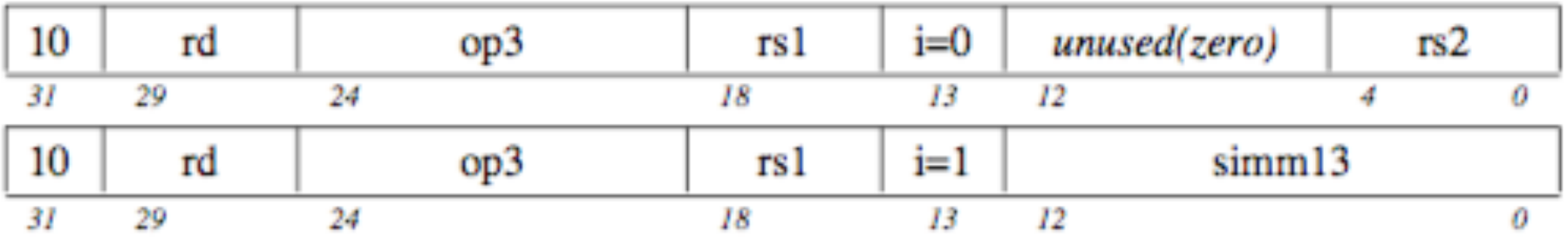
Format 3 (*op* = 2 or 3): Remaining instructions

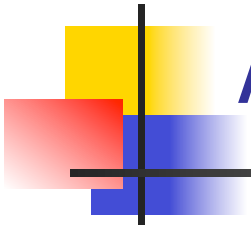


Add format

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):

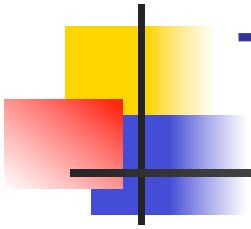




Addi

- r[rd]=r[rs1]+sign_ext(simm13)

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



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