Questions on Lab1

- Hex text file for instructions
  - minmax.imi:
    1a
    20
    9a
    ...

- Hex Text file for data
  - minmax.dmi
    07
    06
    de
    ad
Lab Due Dates

- Lab 1 Due Thursday, July 10 by 5 PM
  - Submit reports to Anjum
- Lab 2 Due Tuesday, July 15 6 PM in class
- No late submissions!

- I plan to disclose a range of clock cycles from your Lab 2 reports (no names)

Example: Saturating Add

- Consider chars a[0] and a[1]
- a[0] and a[1] guaranteed to be positive
  - bit 7 is zero for both a[0] and a[1]
- Saturating add:
  - Add the two numbers. If there is overflow, the result is the largest positive 8-bit number (0x7f)
- Saturating add is used for blending two pixels
Example Data

• Example 1:
  a[0] = 0x17
  a[1] = 0x23
  result = 0x3a, No need to saturate result

• Example 2:
  a[0] = 0x70
  a[1] = 0x39
  result = 0xa9  Need to saturate: result = 0x7f

ISA: Resources

• 4 8-bit General purpose registers:
  – R0, R1, R2, R3

• Special registers
  – Program Counter, 8-bits
  – flag C, 1 bit (condition code)
ISA: Instructions

- Instructions (make a table):
  Rd: destination register, Rs: source register
  $: immediate (constant)

  mov    Rd, $const4 // load 4-bit const
  ld     Rd, *Rs    // load from memory
  add    Rd, Rs     // add registers
  add    Rd, $const2 // add 2-bit const
  sub    Rd, $const2 // subtract 2-bit const
  cmplt  Rd, $const2 // compare less than, set C
  usrl   Rd          // unsigned shift right 1
  jmpF   *Rd         // jump if F=false

Instruction Description

Instruction: mov

Encoding:

| 00 | Rd | 4-bit Constant |

Syntax: mov  Rd, $Constant

Operation: Rd = Constant

Description: 4-bit constant is loaded into register Rd. Upper 4 bits are set to zero (the constant is zero extended).

Flag: The flag C retains it’s old value
Example Assembly Code

- Saturating add

    // sadd.s
    // Register usage: R0: result, R1: memory address, temporary

    0x0c 000 start: mov R1, $0   // R1 = 0, has address of a[0]
    0xe2 001   ld R0, *R1       // R0 = a[0]
    0xa5 002   add R1, $1      // R1 = 1, has address of a[1]
    0xe6 003   ld R1, *R1      // R1 = a[1]
    0xa1 004   add R0, R1      // R0 = R0 + R1
    0x90 005   cmplt R0, $0    // if( R0 < 0 ) C = true
    0x0b 006   mov R1, $0xb     // R1 = 0xb, C unchanged!
    0x21 007   jmpF *R1        // we're done if flag C == false
    0x08 008   mov R0, $0      // R0 = 0
    0xff 009   sub R0, $1      // R0 = 0xff
    0xfe 00a   usrl R0         // R0 = R0 >> 1 = 0x7f
    0x10 00b done: halt        // done!

Microprocessor Design Steps

- Design Instruction Set Architecture (ISA)
- Develop software generation tools
- Code applications
  - Develop instruction set simulator (ISS)
  - Design datapath, verify it
  - Design the Processor, simulate logic
  - Verify the processor
  - Fabricate the chip
Software Generation Steps

- Write program in high level language (C, C++, Java, ...)
  - Compile the program
    - Generates object file
- Write program in assembly language
  - Assemble the program
    - Generates object file
- Link the program
  - Library functions are linked
  - External references are resolved
  - Executable is generated

Format of an executable file

- There are various standards: coff, elf, ...
- Sections
  - Header: describes size and position of other pieces
  - Text: machines code for instructions
  - Data: binary representation of data in source file
  - Relocation information related to absolute addresses
  - Symbol table associated with external labels
  - Debugging information
Loading the Program

- Operations Performed
  - Read the header
  - Create address space for the program
  - Load I-Mem
  - Initialize D-Mem
  - Copy arguments on stack
  - Initialize machine registers
  - Jump to start up routine, copy arguments to registers
  - call main

Are your 3 programs functionally correct?
General Architecture Challenges

• ISA
  – What is the performance?
    • How to find it out for huge programs?
  – How to optimize the ISA?
    • How to test “what if” scenarios?

• Design
  – How to verify the design?
  – What is the golden reference for behavior?

Answer:
Instruction Set Simulator (ISS)
What is an ISS?
• Software model of your processor
• Runs on a PC or workstation
• Is aware of:
  – Processor’s internal resources
  – ISA
  – Size and characteristics of I-Mem & D-Mem
• Provides
  – Run-time statistics
  – Debug capabilities

What does an ISS do?
• Input:
  – Instruction and data memory image
• Output:
  – Instruction trace
  – Run-time statistics
    • Number of instructions executed
    • Utilization of resources and instructions
    • ...
What is a Reset?

- What happens on PC when you hit
  - Power button (cold boot)
    - Does extensive Power On Self Test (POST)
  - Reset button (warm boot)
- Reset is a special pin on a processor
- Processor and hardware is initialized to a known state
  - PC set to a known value
  - Internal registers set to a known value
  - Instruction execution starts

Structure of ISS

- Written in C++ or Java (C is ok too)
- Define a class (say, Proc)
  - Members are resources of the processor
    - PC, register file, I-Mem, D-Mem, …
Methods in ISS

- **loadIMem()**: loads instruction memory image
  - Reads hex machine code from a text file
- **loadDMem()**: loads data memory image
  - Reads hex data from a text file
- **reset()**: reset the processor
  - Initialize internal state of the processor
  - Set PC = 0 and start execution

Methods in ISS: Runloop()

- Executes instructions
- May take an argument
  - Number of instructions to execute
  - Enables single stepping
- Generates instruction trace
  - Disassembles instruction
  - Prints results and other side effects
Components of runloop()

• Fetch:
  – Read instruction from I-Mem at location pointed to by PC

• Decode:
  – Analyze instruction
  – Find opcode and other fields, such as, register(s), immediate data, branch offset, etc.
  – Note destination for the result
  – Read operand(s) from register(s) (if any)

• Execute:
  – Fetch operand(s) from register(s)
  – Do ALU operation
  – Read D-Mem
  – Select result: either ALU output or data read from D-Mem
  – Resolve whether a branch is taken or not
Components of runloop()

- Memory/Register Writeback
  - Write result to register/D-Mem

- Set next PC
  - If instruction is not a control transfer instruction, or branch is not taken
    - PC = PC +1
  - Else
    - PC = target of the branch

Components of runloop()

- Generate instruction trace
  - Disassemble instruction
  - Print instruction side effects

- Stop execution if HALT instruction
- Else continue with the next instruction
Example Instruction Trace

0x5d 009: ADD R3, $0x1 { Result = 0x 3 }
0xfb 010: ST R2, *R3 { Addr = 0x3, Result = 0x15 }
0x18 011: MOV R2, $0x4
0x65 012: SUB R1, $0x { Result = 0x 6 }
0x74 013: CMPEQ R1, $0x0 { Flag = 0 }
0x33 014: MOV R0, $0xf
0x53 015: ADD R0, $0x3 { Result = 0x12 }
0x51 016: ADD R0, $0x1 { Result = 0x13 }
0x40 017: JMPF *R0
0xf6 019: ST R1, *R2 { Addr = 0x4, Result = 0x6 }
0x19 020: MOV R2, $0x5

Utility Methods in ISS

• dumpIMem():
  – Dumps contents of instruction memory
  – Optionally takes start and end address

• dumpDMem():
  – Dumps contents of data memory
  – Optionally takes start and end address

• Both useful for debugging the program
Main()

• Instantiate class Proc
• Load I-Mem, D-Mem
• Reset the processor
• Interactive loop
  – Receive user command
  – Execute runloop()
  – Dump memory

What you will turn in for Lab 2

• Summary of your ISA from Lab 1 as well as assembly listing for the three programs, including the machine code for each instruction.
• Complete code for the ISS, including all the header files and makefile.
• Complete instruction trace for the three programs executing correctly on the ISS.
• Data memory dump before and after running each one of the three programs. Also, dynamic instruction count for each one of the three programs.
• Answer to the questions below.
• Use “turnin” script (instructions are on the web)
Share your experience

• What did you learn?
• Importance of developing ISS
• Be honest: no penalty if you report problems

Questions

• Were there any bugs in your machine code related to the manual conversion of the assembly code? How would you avoid such bugs?
• How did you code the target addresses for branches? How did you update the branch offsets when the target address changed due to addition or deletion of instruction(s)?
• Were your assembly programs bug free? What was the nature of the bugs in your programs that you uncovered while running on the ISS?
• Did you uncover any deficiencies in the ISA or internal resources of your processor while debugging the programs? How did you fix them?
• What is the dynamic instruction count for each one of the three programs? How do these numbers compare with the manually determined numbers reported by you in Lab 1?
• Based on your experience, give at least three advantages of developing an ISS while designing a new processor architecture.
Tips for debugging programs

- Typical problems
  - Machine code translation errors
  - ISS bugs
  - Logic errors in your program code
  - Infinite loops
  - Branch to invalid I-Mem location
  - Memory fetch from invalid D-Mem location
  - Branch target may change if you modify code

- Print instruction side effects in trace!

You should have completely functional three programs at the end of Lab2!
Homework: Review LogicWorks 4

- Head start on Lab 3 where:
  - You will design datapath
    - Register file, ALU and other elements
    - Implement and test logic
- No submission!

LogicWorks 4 Review

- Various components available in Standard Libraries.
  - D Flip-flop, logic gates, multiplexors, registers, adders, clock, binary switch, binary display, hex keypad, hex display, etc.
- How to connect a bus to various components.
- How to define a subcircuit bottom up
  - Create a subcircuit
  - Use it to define the pins on the parent symbol.
- How to simulate a circuit and generate waveforms.
What you need for Lab2

- UNIX workstations
  - Accounts
- Tools
  - GCC
  - GDB or DDD debugger

Lab Due Dates

- Lab 1 Due Thursday, July 10 by 5 PM
  - Submit to Anjum
- Lab 2 Due Tuesday, July 15 6 PM in class
- No late submissions!