Lab 2 preview

Hung-Wei Tseng
Form your group!

• Form your group!
  • We need 2 persons to make a processor
  • 4 or 5 are too more, we are not making burgers
In Lab 2...

• You will be implementing datapath elements for a single cycle processor
  • We provide ALU, data memory, instruction
  • You need to implement the rest
• Figure out the schematic of your processor first
  • What are the elements required for a single cycle processor
  • How to test each component
  • How to connect them together
Basic steps for executing instructions

- **Instruction fetch**: where?

- **Decode**:
  - What’s the instruction?
  - Where are the operands? **registers**

- **Execute** **ALUs**

- **Memory access** **data memory**
  - Where is my data?

- **Write back** **registers**
  - Where to put the result

- **Determine the next PC**
Build the processor step by step
Basic steps for executing instructions

- Instruction fetch: where?

Instruction memory

- Decode:
  - What's the instruction?
  - Where are the operands?

Execute

- Memory access
  - Where is my data?

Write back

- Where to put the result

Determine the next PC
The datapath

Instruction Memory

Instruction Address [31:0]

Instruction [31:0]

ADD

PC

4
Basic steps for executing instructions

- Instruction fetch: where?

- Decode:
  - What’s the instruction?
  - Where are the operands? *registers*

```plaintext
120007a30: 0f00bb27  ldah gp,15(t12)
120007a34: 509cbd23  lda   gp,-25520(gp)
120007a38: 00005d24  ldah  t1,0(gp)
120007a3c: 0000bd24  ldah  t4,0(gp)
120007a40: 2ca422a0  ldl   t0,-23508(t1)
120007a44: 130020e4  beq   t0,120007a94
120007a48: 00003d24  ldah  t0,0(gp)
120007a4c: 2ca4e2b3  stl   zero,-23508(t1)
```

Instruction memory

**ALU**

**Processor**

**Memory access**

**Write back**

**Determine the next PC**

**800bf9000: 00c2e800 12773376**

**800bf9004: 00000008 8**

**800bf9008: 00c2f000 12775424**

**800bf900c: 00000008 8**

**800bf9010: 00c2f800 12777472**

**800bf9014: 00000008 8**

**800bf9018: 00c30000 12779520**

**800bf901c: 00000008 8**

8
R-type

- op $rd, $rs, $rt
  - 3 regs.: add, addu, and, nor, or, sltu, sub, subu
  - 2 regs.: sll, srl
  - 1 reg.: jr
- 1 arithmetic operation, 1 I-memory access
- Example:
    opcode = 0x0, funct = 0x20
  - sll $t0, $t1, 8: R[8] = R[9] << 8
    opcode = 0x0, shamt = 0x8, funct = 0x0
I-type

- \( \text{op } $rt, $rs, \text{immediate} \)
  - \( \text{addi, addiu, andi, beq, bne, ori, slti, sltiu} \)
- \( \text{op } $rt, \text{offset($rs)} \)
  - \( \text{lw, lbu, lhu, ll, lui, sw, sb, sc, sh} \)
- 1 \text{ arithmetic op, 1 } I\text{-memory and 1 } D\text{-memory access}
- Example:
  - \( \text{lw } $s0, 4($s2) \)
    \( R[16] = \text{mem}[R[18]+4] \)
  - \( \text{lw } $s0, 0($s2) \)
  - \( \text{add } $s2, $s2, $s1 \)

- Table:
  - | opcode | rs | rt | immediate |
    |-------|----|----|-----------|
    | 6 bits | 5 bits | 5 bits | 16 bits |
- **I-type (cont.)**

<table>
<thead>
<tr>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
</tr>
</tbody>
</table>

- **op $rt, $rs, immediate**
  - addi, addiu, andi, beq, bne, ori, slti, sltiu
- **op $rt, offset($rs)**
  - lw, lbu, lhu, ll, lui, sw, sb, sc, sh
- **1 arithmetic op, 1 I-memory and 1 D-memory access**
- **Example:**
  - `beq $t0, $t1, -40`
  - `if (R[8] == R[9]) PC = PC + 4 + 4*(-40)`
J-type

- op immediate
  - j, jal
- 1 instruction memory access, 1 arithmetic op
- Example:
  - jal quicksort
    \[ R[31] = PC + 4 \]
    \[ PC = \text{quicksort} \]
The datapath

Instruction Memory
- Instruction Address [31:0]
- Instruction [31:0]

Register File
- Read Reg 1
- Read Data 1
- Read Reg 2
- Read Data 2
- Write Reg
- Write Data

Instruction
- [25-21]
- [20-16]
- [15-11]
- [15-0]

PC

sign-extend

ADD

6 bits  5 bits  5 bits  5 bits  5 bits  6 bits
opcode  rs  rt  rd  shift amount  funct

6 bits  5 bits  5 bits  16 bits
opcode  rs  rt  immediate

6 bits  5 bits  26 bits
opcode  immediate
The datapath
The datapath
The datapath
You have to implement the processor.

You have to implement the red parts!
module inst_rom(
    input clock,
    input reset,

    input [31:0] addr_in, //Connect to PC_next
    output [31:0] data_out //Fetched instruction
);
parameter INIT_PROGRAM="c:/myfiles/blank.memh";
endmodule
Interface of ALU

module alu(
    input [5:0] Func_in,
    input [31:0] A_in,
    input [31:0] B_in,
    output [31:0] O_out,
    output Branch_out,
    output Jump_out
);

module data_memory(
    input clock,
    input reset,
    input [31:0] addr_in, //Read/Write address
    input [31:0] writedata_in, //Data to write to memory
    input re_in, //Read Enable – set high when reading from memory
    input we_in, //Write Enable – set high when writing to memory
    output [31:0] readdata_out, //Data output for reads from memory
    input [1:0] size_in, //Not used yet – hardwire to 2'b11
    input [7:0] serial_in,
    input serial_ready_in,
    input serial_valid_in,
    output [7:0] serial_out,
    output serial_rden_out,
    output serial_wden_out
);

parameter INIT_PROGRAM0="c:/myfiles/blank.memh";
parameter INIT_PROGRAM1="c:/myfiles/blank.memh";
parameter INIT_PROGRAM2="c:/myfiles/blank.memh";
parameter INIT_PROGRAM3="c:/myfiles/blank.memh";
Your register file design

- Should contain at least the above inputs/outputs
  - Does every instruction write back data?
- Is it clocked?
- Support reset
- Deal with $zero
When you interview with us

• Show us it’s working
• Show us your processor.v
• Show us your register_file.v
  • Demonstrate it works: you need to design a testbench for it.
• Show us your schematic
  • You may output it through the RTL viewer under analysis tool in Quartus II
Announcement

- Lecture next Wednesday for Lab 3
- Lab #1 due this Friday
Q & A