

## CS61C : Machine Structures

### Lecture 5.1.1

### CPU Design I

2004-07-19

Kurt Meinz

inst.eecs.berkeley.edu/~cs61c



CS 61C L5.1.1 CPU Design I (1)

K. Meinz, Summer 2004 © UCB

## Review: Verilog Dataflow Paradigm

```
module and_or(Z, A, B, Sel);  
  input A, B, Sel;  
  output Z;  
  wire Z;  
  assign #5 Z = Sel ? (A & B) : (A | B);  
endmodule
```

### • Continuous Assignment:

- Limited to simple operations
  - Simple translation to structural
  - Trinary if ( ? : ), logic operators
- Excellent for CL.



CS 61C L5.1.1 CPU Design I (2)

K. Meinz, Summer 2004 © UCB

## Review: What's a reg??

### • Output types:

- Structural → Wire
  - wire a,b,c; and (a, b, c);
- Continuous Assign → Wire
  - wire a,b,c; assign a = b & c;
- Procedural Assign → **REG**
  - wire b,c; reg a; always @ (b or c) a = b & c;

### • Why?

- Structural, continuous always function of current input. → simple wire
- REG ↔ Verilog has to remember value



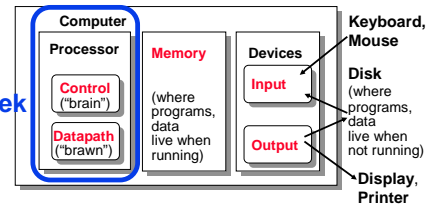
CS 61C L5.1.1 CPU Design I (3)

K. Meinz, Summer 2004 © UCB

## Anatomy: 5 components of any Computer



This week



CS 61C L5.1.1 CPU Design I (4)

K. Meinz, Summer 2004 © UCB

## Outline

- Design a processor: step-by-step
- Requirements of the Instruction Set
- Hardware components that match the instruction set requirements



CS 61C L5.1.1 CPU Design I (5)

K. Meinz, Summer 2004 © UCB

## How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) => datapath **requirements**
  - meaning of each instruction is given by the **register transfers**
  - datapath must include storage element for ISA registers
  - datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. **Assemble** datapath meeting requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic



CS 61C L5.1.1 CPU Design I (6)

K. Meinz, Summer 2004 © UCB



## Step 2a: Components of the Datapath

- **Combinational Elements**
- **Storage Elements**
  - Clocking methodology



CS 61C L5.1.1 CPU Design I (13)

K. Meitz, Summer 2004 © UCB

## Combinational Logic: 16-bit Sign Extender

```
//Sign extender from 16- to 32-bits.
module signExtend (in,out);
  input  [15:0] in;
  output [31:0] out;
  reg    [31:0] out;

  assign
  out = { in[15], in[15], in[15], in[15],
          in[15], in[15], in[15], in[15],
          in[15], in[15], in[15], in[15],
          in[15], in[15], in[15], in[15],
          in[15:0] };
endmodule // signExtend
```



CS 61C L5.1.1 CPU Design I (14)

K. Meitz, Summer 2004 © UCB

## Combinational Logic: 2-bit Left Shift

```
// 32-bit Shift left by 2
module leftShift2 (in,out);
  input [31:0] in;
  output [31:0] out;

  assign
  out = { in[29:0], 1'b0, 1'b0 };
endmodule // leftShift2

(Also: assign out = in[29:0] << 2;)
```

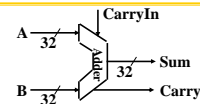


CS 61C L5.1.1 CPU Design I (15)

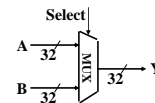
K. Meitz, Summer 2004 © UCB

## Combinational Logic: More Elements

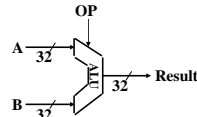
### • Adder



### • MUX



### • ALU



CS 61C L5.1.1 CPU Design I (16)

K. Meitz, Summer 2004 © UCB

## Combinational Logic: 32-bit Adder

```
//Behavioral model of 32-bit adder.
module add32 (S,A,B);
  input  [31:0] A,B; //32 bits
  output [31:0] S;
  reg    [31:0] S;

  always @ (A or B)
    S = A + B;
endmodule // add32
```



CS 61C L5.1.1 CPU Design I (17)

K. Meitz, Summer 2004 © UCB

How do we make this datapath?

## Combinational Logic: 32-bit Mux

```
// Behavioral model of 32-bit wide
// 2-to-1 multiplexor.
module mux32 (in0,in1,select,out);
  input [31:0] in0,in1;
  input  select;
  output [31:0] out;

  reg [31:0] out;
  always @ (in0 or in1 or select)
    if (select) out=in1;
    else out=in0;

endmodule // mux32
```



CS 61C L5.1.1 CPU Design I (18)

K. Meitz, Summer 2004 © UCB

### CL: ALU for MIPS-lite (1/4)

- Addition, subtraction, logical OR, ==:

```
ADDU R[rd] = R[rs] + R[rt]; ...
```

```
SUBU R[rd] = R[rs] - R[rt]; ...
```

```
ORI R[rt] = R[rs] | zero_ext(Imm16)...
```

```
BEQ if ( R[rs] == R[rt] ) ...
```

- Test to see if output == 0 for any ALU operation gives == test. How?

- P&H Section 4.5 also adds AND, Set Less Than (1 if A < B, 0 otherwise)

- Behavioral ALU follows sec 4.5



CS 61C L5.1.1 CPU Design I (19)

K. Meina, Summer 2004 © UCB

### CL: ALU (2/4)

```
//Behavioral model of ALU:  
// 8 functions and "zero" flag,  
// A is top input, B is bottom,  
// according to P&H figure 5.19.
```

```
module ALU (A,B,control,zero,result);  
input [31:0] A, B;  
input [2:0] control;  
output zero; // used for beq,bne  
output [31:0] result;
```

```
reg [31:0] result, temp;  
always @ (A or B or control)...
```



CS 61C L5.1.1 CPU Design I (20)

K. Meina, Summer 2004 © UCB

### CL: ALU (3/4)

```
reg [31:0] result, C;  
always @ (A or B or control)  
begin  
case (control)  
3'b000: // AND  
result=A&B;  
3'b001: // OR  
result=A|B;  
3'b010: // add  
result=A+B;  
3'b110: // subtract  
result=A-B;  
3'b111: // slt  
result=(A<B)? 1 : 0;  
endcase // case(control)  
end // always @ (A or B or control)  
assign zero = (result==0);  
endmodule // ALU
```



CS 61C L5.1.1 CPU Design I (21)

K. Meina, Summer 2004 © UCB

©Unsigned compare! Can you fix it?

### CL: ALU (4/4)

```
// if A and B have the same sign,  
// then A<B works(slt == 1 if A-B<0)  
// if A and B have different signs,  
// then A<B if A is negative  
// (slt == 1 if A<0)
```

```
...  
3'b111: // slt  
begin  
temp = A - B;  
result = (A[31]^B[31]) ?  
A[31] :  
temp[31];  
end  
...
```



CS 61C L5.1.1 CPU Design I (22)

K. Meina, Summer 2004 © UCB

### Step 2b: Components of the Datapath

- Combinational Elements

- Storage Elements

- Clocking methodology



CS 61C L5.1.1 CPU Design I (23)

K. Meina, Summer 2004 © UCB

### Storage Element: Idealized Memory

- Memory (idealized)

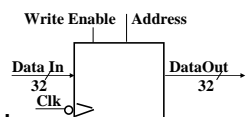
- One input bus: Data In
- One output bus: Data Out

- Memory word is selected by:

- Address selects the word to put on Data Out
- Write Enable = 1: address selects the memory word to be written via the Data In bus

- Clock input (CLK)

- The CLK input is a factor ONLY during write operation
- During read operation, behaves as a combinational logic block:
  - Address valid => Data Out valid after "access time."



CS 61C L5.1.1 CPU Design I (24)

K. Meina, Summer 2004 © UCB

### Verilog Memory for MIPS Interpreter (1/3)

```
// Behavioral model of Random Access Memory:
// 32-bit wide, 256 words deep,
// asynchronous read-port if RD=1,
// synchronous write-port if WR=1,
// initialize from hex file ("data.dat")
// on positive edge of reset signal,
// dump to binary file ("dump.dat")
// on positive edge of dump signal.
module mem
  (CLK,RST,DMP,WR,RD,address,writeD,readD);
  input CLK, RST, DMP, WR, RD;
  input [31:0] address, writeD;
  output [31:0] readD;
  reg [31:0] readD;
  parameter memSize=256;
  reg [31:0] memArray [0:memSize-1];
  integer chann,i;
```



### Verilog Memory for MIPS Interpreter (2/3)

```
integer chann,i;
always @ (posedge RST)
  $readmemh("data.dat", memArray);

// write if WR & positive clock edge (synchronous)
always @ (posedge CLK)
  if (WR) memArray[address[9:2]] =
    writeD;

// read if RD, independent of clock (asynchronous)
always @ (address or RD)*
  if (RD)
    readD = memArray[address[9:2]];
endmodule
```



Ⓜ See how sneaky sensitivity lists can be!  
Use an assign!

### Why is it "memArray[address[9:2]]"?

- Our memory is always byte-addressed
  - We can lb from 0x0, 0x1, 0x2, 0x3, ...
- lw only reads word-aligned requests
  - We only call lw with 0x0, 0x4, 0x8, 0xC, ...
  - I.e., the last two bits are always 0
- memArray is a word wide and 2<sup>8</sup> deep
  - reg [31:0] memArray [0:256-1];
  - Size = 4 Bytes/row \* 256 rows = 1024 B
  - If we're simulating lw/sw, we R/W words
  - What bits select the first 256 words? [9:2]!
  - 1<sup>st</sup> word = 0x0 = 0b000 = memArray[0];
  - 2<sup>nd</sup> word = 0x4 = 0b100 = memArray[1], etc.



### Verilog Memory for MIPS Interpreter (3/3)

```
end;
always @ (posedge DMP)
  begin
    chann = $fopen("dump.dat");
    if (chann==0)
      begin
        $display("$fopen of dump.dat
failed.");
        $finish;
      end
    // Temp variables chan, i
    for (i=0; i<memSize; i=i+1)
      begin
        $fdisplay(chann, "%b",
          memArray[i]);
      end
    end // always @ (posedge DMP)
endmodule // mem
```

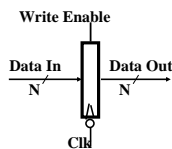


### Storage Element: Register (Building Block)

#### • 32-bit Register

- Similar to the D Flip Flop except

- N-bit input and output
- Write Enable input (CE)



#### • Write Enable:

- negated (or deasserted) (0):  
Data Out will not change
- asserted (1):  
Data Out will become Data In



### Verilog 32-bit Register

```
// Behavioral model of 32-bit Register:
// positive edge-triggered,
// synchronous active-high reset.
module reg32 (CLK,Q,D,RST);
  input [31:0] D;
  input CLK, RST;
  output [31:0] Q;

  reg [31:0] Q;
  always @ (posedge CLK)
    if (RST) Q = 0; else Q = D;
endmodule // reg32
```

