Introduction to CMOS VLSI Design

Lecture 2: MIPS Processor Example

David Harris

Harvey Mudd College
Spring 2004

Outline

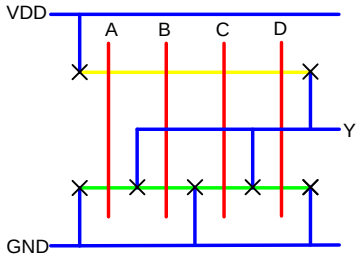
- Design Partitioning
- MIPS Processor Example
  - Architecture
  - Microarchitecture
  - Logic Design
  - Circuit Design
  - Physical Design
- Fabrication, Packaging, Testing

Activity 2

- Sketch a stick diagram for a 4-input NOR gate

Activity 2

- Sketch a stick diagram for a 4-input NOR gate



Coping with Complexity

- How to design System-on-Chip?
  - Many millions (soon billions!) of transistors
  - Tens to hundreds of engineers
- Structured Design
- Design Partitioning

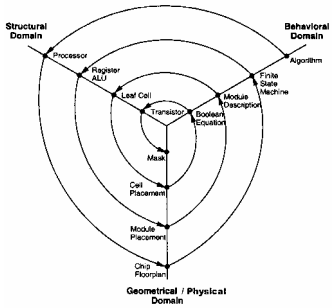
Structured Design

- **Hierarchy**: Divide and Conquer
  - Recursively system into modules
- **Regularity**
  - Reuse modules wherever possible
  - Ex: Standard cell library
- **Modularity**: well-formed interfaces
  - Allows modules to be treated as black boxes
- **Locality**
  - Physical and temporal

Design Partitioning

- **Architecture**: User's perspective, what does it do?
  - Instruction set, registers
  - MIPS, x86, Alpha, PIC, ARM, …
- **Microarchitecture**
  - Single cycle, multcycle, pipelined, superscalar?
- **Logic**: how are functional blocks constructed
  - Ripple carry, carry lookahead, carry select adders
- **Circuit**: how are transistors used
  - Complementary CMOS, pass transistors, domino
- **Physical**: chip layout
  - Datapaths, memories, random logic

Gajski Y-Chart



MIPS Architecture

- Example: subset of MIPS processor architecture
  - Drawn from Patterson & Hennessy
- MIPS is a 32-bit architecture with 32 registers
  - Consider 8-bit subset using 8-bit datapath
  - Only implement 8 registers ($0 - $7)
  - $0 hardwired to 00000000
  - 8-bit program counter
- You'll build this processor in the labs
  - Illustrate the key concepts in VLSI design

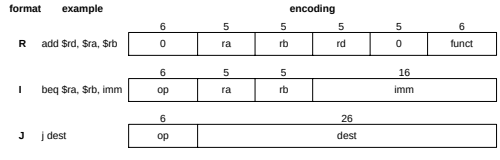
Instruction Set

Table 1.7 MIPS instruction set (subset supported)

Instruction	Function		Encoding	op	funct
add $1, $2, $3	addition:	$1 ← $2 + $3	R	000000	100000
sub $1, $2, $3	subtraction:	$1 ← $2 – $3	R	000000	100010
and $1, $2, $3	bitwise and:	$1 ← $2 and $3	R	000000	100100
or $1, $2, $3	bitwise or:	$1 ← $2 or $3	R	000000	100101
slt $1, $2, $3	set less than:	$1 ← 1 if $2 < $3 $1 ← 0 otherwise	R	000000	101010
addi $1, $2,	add immediate:	$1← $2 + imm	I	001000	n/a
beq $1, $2, imm	branch if equal:	PC ← PC + imm[a]	I	000100	n/a
j destination	jump:	PC_destination[a]	J	000010	n/a
lb $1, imm($2)	load byte:	$1 ← mem[$2 + imm]	I	100000	n/a
sb $1, imm($2)	store byte:	mem[$2 + imm] ← $1	I	110000	n/a

Instruction Encoding

- 32-bit instruction encoding
  - Requires four cycles to fetch on 8-bit datapath



Fibonacci (C)

$f_0 = 1; f_{-1} = -1$

$f_n = f_{n-1} + f_{n-2}$

f = 1, 1, 2, 3, 5, 8, 13, …

```
int fib(void)
{
  int n = 8;            /* compute nth Fibonacci number */
  int f1 = 1, f2 = -1; /* last two Fibonacci numbers */

  while (n != 0) {      /* count down to n = 0 */
    f1 = f1 + f2;
    f2 = f1 - f2;
    n = n - 1;
  }
  return f1;
}
```

Fibonacci (Assembly)

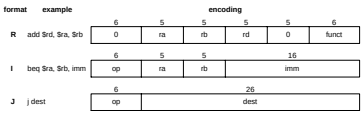
- 1st statement: n = 8
- How do we translate this to assembly?

Fibonacci (Assembly)

```
# fib.asm
# Register usage: $3: n $4: f1 $5: f2
# return value written to address 255
fib:  addi $3, $0, 8      # initialize n=8
      addi $4, $0, 1      # initialize f1 = 1
      addi $5, $0, -1     # initialize f2 = -1
loop: beq $3, $0, end     # Done with loop if n = 0
      add $4, $4, $5      # f1 = f1 + f2
      sub $5, $4, $5      # f2 = f1 - f2
      addi $3, $3, -1     # n = n - 1
      j loop              # repeat until done
end: sb $4, 255($0)       # store result in address 255
```


Fibonacci (Binary)

- 1st statement: addi $3, $0, 8
- How do we translate this to machine language?
  - Hint: use instruction encodings below



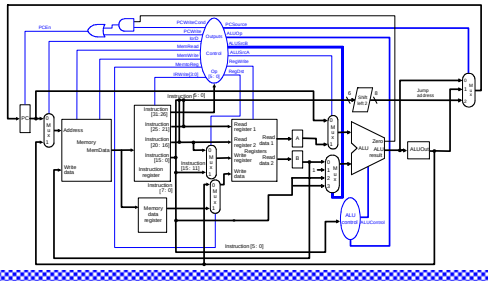
Fibonacci (Binary)

- Machine language program

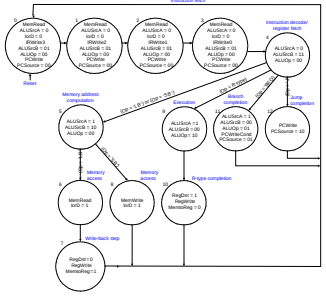
Instruction	Binary Encoding			Hexadecimal Encoding
addi $3, $0, 8	001000	00000	00011	0000000000001000	20030008
addi $4, $0, 1	001000	00000	00100	0000000000000001	20040001
addi $5, $0, -1	001000	00000	00101	1111111111111111	2005ffff
beq $3, $0, end	000100	00011	00000	0000000000000101	10600005
add $4, $4, $5	000000	00100	00101	00100	00000	100000	00852020
sub $5, $4, $5	000000	00100	00101	00101	00000	100010	00852822
addi $3, $3, -1	001000	00011	00011	1111111111111111	2063ffff
j loop	000010	00000000000000000000000000000011	08000003
sb $4, 255($0)	110000	00000	00100	0000000011111111	a00400ff

MIPS Microarchitecture

- Multicycle μarchitecture from Patterson & Hennessy

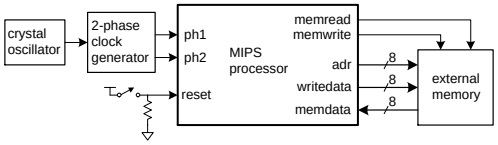


Multicycle Controller

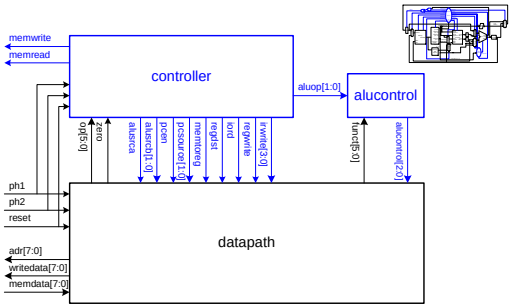


Logic Design

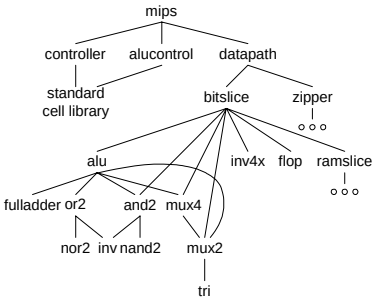
- Start at top level
  - Hierarchically decompose MIPS into units
- Top-level interface



Block Diagram



Hierarchical Design



HDLs

- Hardware Description Languages
  - Widely used in logic design
  - Verilog and VHDL
- Describe hardware using code
  - Document logic functions
  - Simulate logic before building
  - Synthesize code into gates and layout
    - Requires a library of standard cells

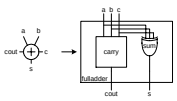
Verilog Example

```
module fulladder(input  a, b, c,
                 output s, cout);

sum         s1(a, b, c, s);
carry       c1(a, b, c, cout);
endmodule

module carry(input  a, b, c,
             output cout)

assign cout = (a&b) | (a&c) | (b&c);
endmodule
```



Circuit Design

- How should logic be implemented?
  - NANDs and NORs vs. ANDs and ORs?
  - Fan-in and fan-out?
  - How wide should transistors be?
- These choices affect speed, area, power
- Logic synthesis makes these choices for you
  - Good enough for many applications
  - Hand-crafted circuits are still better

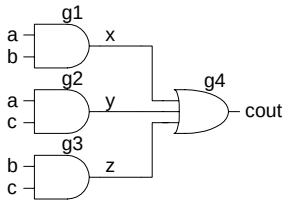
Example: Carry Logic

- `assign cout = (a&b) | (a&c) | (b&c);`

Transistors? Gate Delays?

Example: Carry Logic

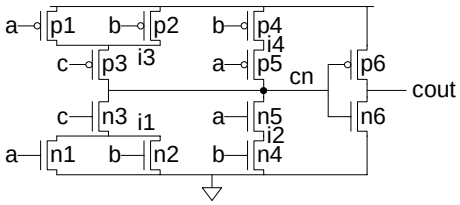
- `assign cout = (a&b) | (a&c) | (b&c);`



Transistors? Gate Delays?

Example: Carry Logic

- `assign cout = (a&b) | (a&c) | (b&c);`

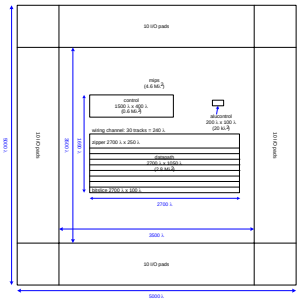


Transistors? Gate Delays?

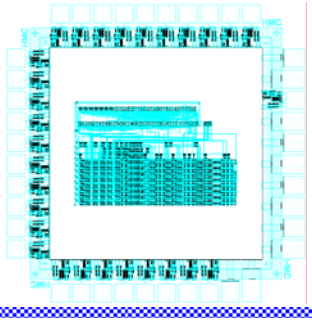
Physical Design

- Floorplan
- Standard cells
  - Place & route
- Datapaths
  - Slice planning
- Area estimation

MIPS Floorplan

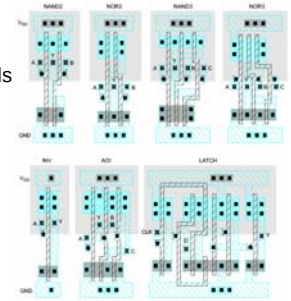


MIPS Layout



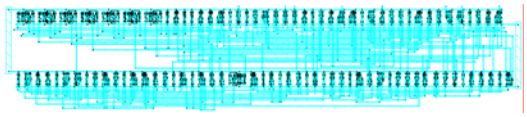
Standard Cells

- Uniform cell height
- Uniform well height
- M1 V_{DD} and GND rails
- M2 Access to I/Os
- Well / substrate taps
- Exploits regularity



Synthesized Controller

- Synthesize HDL into gate-level netlist
- Place & Route using standard cell library



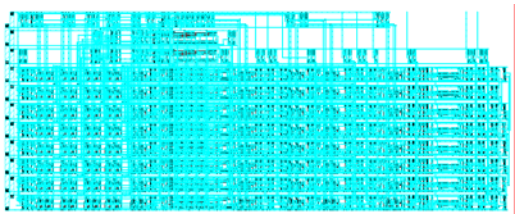
Pitch Matching

- Synthesized controller area is mostly wires
  - Design is smaller if wires run through/over cells
  - Smaller = faster, lower power as well!
- Design snap-together cells for datapaths and arrays
  - Plan wires into cells
  - Connect by abutment
    - Exploits locality
    - Takes lots of effort

A	A	A	A	B
A	A	A	A	B
A	A	A	A	B
A	A	A	A	B
C		C		D

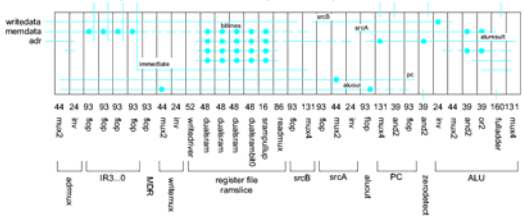
MIPS Datapath

- 8-bit datapath built from 8 bitslices (regularity)
- Zipper at top drives control signals to datapath



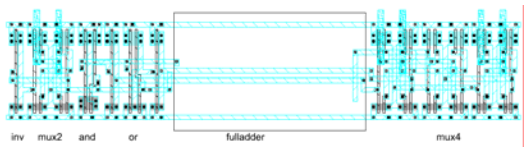
Slice Plans

- Slice plan for bitslice
  - Cell ordering, dimensions, wiring tracks
  - Arrange cells for wiring locality



MIPS ALU

- Arithmetic / Logic Unit is part of bitslice



Area Estimation

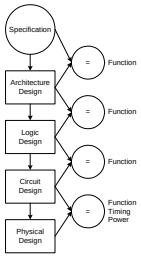
- Need area estimates to make floorplan
  - Compare to another block you already designed
  - Or estimate from transistor counts
  - Budget room for large wiring tracks
  - Your mileage may vary!

Table 1.10 Typical layout densities

Element	Area
random logic (2-level metal process)	1000 – 1500 λ^2 / transistor
datapath	250 – 750 λ^2 / transistor or 6 WL + 360 λ^2 / transistor
SRAM	1000 λ^2 / bit
DRAM (in a DRAM process)	100 λ^2 / bit
ROM	100 λ^2 / bit

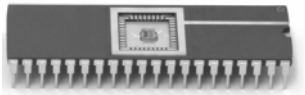
Design Verification

- Fabrication is slow & expensive
  - MOSIS 0.6μm: $1000, 3 months
  - State of art: $1M, 1 month
- Debugging chips is very hard
  - Limited visibility into operation
- Prove design is right before building!
  - Logic simulation
  - Ckt. simulation / formal verification
  - Layout vs. schematic comparison
  - Design & electrical rule checks
- Verification is > 50% of effort on most chips!



Fabrication & Packaging

- Tapeout final layout
- Fabrication
  - 6, 8, 12” wafers
  - Optimized for throughput, not latency (10 weeks!)
  - Cut into individual dice
- Packaging
  - Bond gold wires from die I/O pads to package



Testing

- Test that chip operates
  - Design errors
  - Manufacturing errors
- A single dust particle or wafer defect kills a die
  - Yields from 90% to < 10%
  - Depends on die size, maturity of process
  - Test each part before shipping to customer

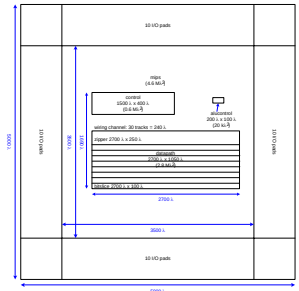
Project Strategy

- Proposal
  - Specifies inputs, outputs, relation between them
- Floorplan
  - Begins with block diagram
  - Annotate dimensions and location of each block
  - Requires detailed paper design
- Schematic
  - Make paper design simulate correctly
- Layout
  - Physical design, DRC, NCC, ERC

Floorplan

- How do you estimate block areas?
  - Begin with block diagram
  - Each block has
    - Inputs
    - Outputs
    - Function (draw schematic)
    - Type: array, datapath, random logic
- Estimation depends on type of logic

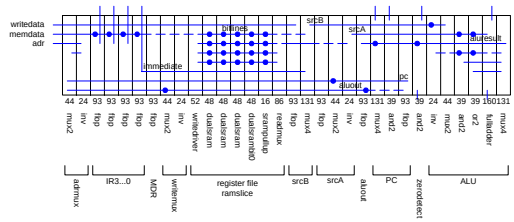
MIPS Floorplan



Area Estimation

- Arrays:
  - Layout basic cell
  - Calculate core area from # of cells
  - Allow area for decoders, column circuitry
- Datapaths
  - Sketch slice plan
  - Count area of cells from cell library
  - Ensure wiring is possible
- Random logic
  - Compare complexity do a design you have done

MIPS Slice Plan



Typical Layout Densities

- Typical numbers of high-quality layout
- Derate by 2 for class projects to allow routing and some sloppy layout.
- Allocate space for big wiring channels

Element	Area
Random logic (2 metal layers)	1000-1500 λ^2 / transistor
Datapath	250 – 750 λ^2 / transistor Or 6 WL + 360 λ^2 / transistor
SRAM	1000 λ^2 / bit
DRAM	100 λ^2 / bit
ROM	100 λ^2 / bit