Lecture #8: Lab 3

EE183 Pipelined Processor

Kunle Olukotun
Stanford EE183

February 3, 2003

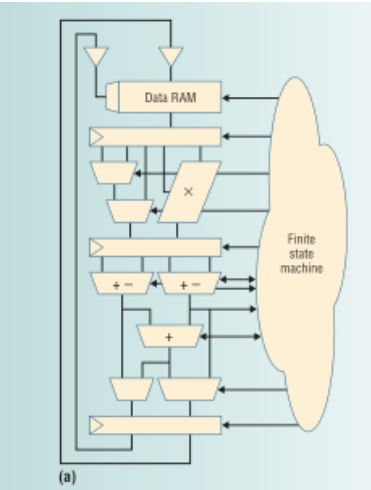
Lab Stuff

- Lab #2 due Friday at 6pm
- I ‘ll be in the lab at 5pm or so for demos.
- Any questions?

System-on-Chip (SoC) Design Challenges

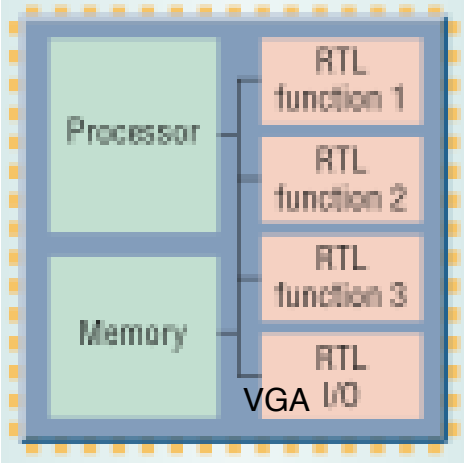
- > 100,000 gates/mm^2
- Verification is a bottleneck
  - Increasing complexity
  - >70% of resources on verification
- Cost of bugs rising
  - 1 million mask cost
  - Extra design time
- Shorter time to market
  - Markets change
  - Standards change
- Need to increase designer productivity

RTL Design



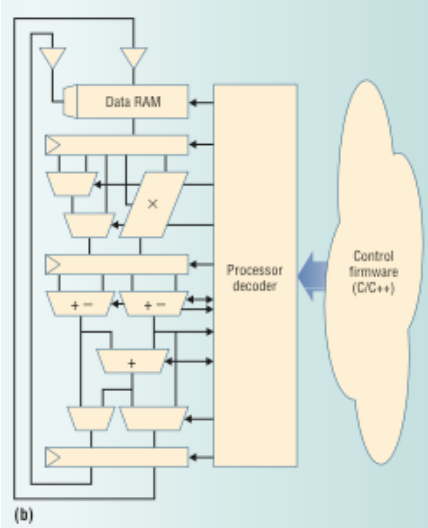
- Datapath
  - Fixed purpose
  - consumes most of gates
- Control
  - Datapath sequencing
  - Error conditions
  - Communication
  - Control consumes most of verification time

Processor Based SOC



- Processors controls hardwired functions
- Memory level communication
  - Memory mapped I/O
  - VGA example
- More software development
- Added flexibility
  - Decreased design time
  - Adapt to changing standards
- Embedded processors
  - MIPS, ARM, PowerPC

Application Specific Instruction-set Processor (ASIP)



- Specialized datapath (FUs)
- Register level communication
- ISA changes
- Advantages
  - Raises design abstraction: C instead of Verilog
  - Eliminates most control logic
  - Added flexibility
- Configurable processors
  - Tensilica
  - ARC
- Final project

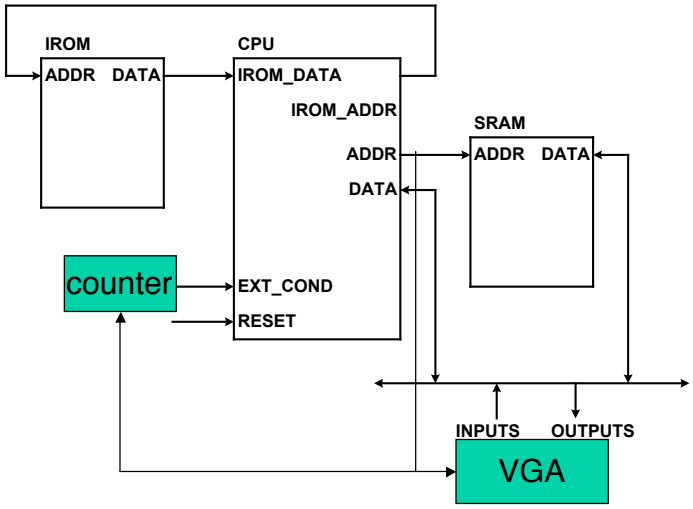
Processor Overview

- 12-bit RISC Microcontroller
  - What would having only 8 bits mean for addressing memory?
- 4 or 8 General Purpose Registers
  - 4 back in the days when we had a small FPGA ☺
- 43 Instructions
- 3 operand instructions
- 4 stage pipeline
- Register indirect addressing mode
  - What does this mean?

Why Design this Processor?

- Complex enough to be "interesting"
- Simple enough to complete in 2 weeks
- Pipelining is an important technique in digital design
- ***Exciting!*** Tell your friends and look cool at dinner parties

Processor Overview

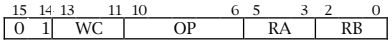


Instruction Set Architecture (ISA)

- 8 General Purpose Registers
- ALU Instructions
  - 28 Instructions
  - 3 operands
- Control Transfer Instructions
  - 12 Instructions
  - Conditional/Unconditional branches
- Memory Instructions
  - 2 instructions
  - Load/Store

ALU Instructions I

ALU Instruction format



Arithmetic

OP_{hex}	operation	mnemonic	
00	C = A + B	`ADD`	*C, A, B*
01	C = A + B + 1	`ADDINC`	*C, A, B*
02	C = A	`PASSA`	*C, A*
03	C = A + 1	`INCA`	*C, A*
04	C = A - B - 1	`SUBDEC`	*C, A, B*
05	C = A - B	`SUB`	*C, A, B*
06	C = A - 1	`DECA`	*C, A*
07	C = A	`PASSA`	*C, A*

ALU Instructions II

Shift Instructions

OP_{hex}	operation	mnemonic	
08	C = Logical Shift Left(A)	`LSL`	*C, A*
09	C = Arith Shift Right(A)	`ASR`	*C, A*

Boolean Instructions

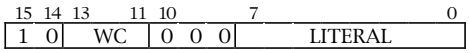
OP_{hex}	operation	mnemonic	
10	C = 0	`ZEROS`	*C*
11	C = A • B	`AND`	*C, A, B*
12	C = A' • B	`ANDNOTA`	*C, A, B*
13	C = B	`PASSB`	*C, B*
14	C = A • B'	`ANDNOTB`	*C, A, B*
15	C = A	`PASSA`	*C, A*
16	C = A ⊕ B	`XOR`	*C, A, B*
17	C = A + B	`OR`	*C, A, B*
18	C = A' • B'	`NOR`	*C, A, B*
19	C = A ⊕ B'	`XNOR`	*C, A, B*
1A	C = A'	`PASSNOTA`	*C, A*
1B	C = A' + B	`ORNOTA`	*C, A, B*
1C	C = B'	`PASSNOTB`	*C, B*
1D	C = A + B'	`ORNOTB`	*C, A, B*
1E	C = A' + B'	`NAND`	*C, A, B*
1F	C = 1	`ONES`	*C*

Literal Instruction

Literal Instruction

OP_{hex}	operation	mnemonic
02	C = literal	`LOADLIT C, literal`

Old literal Instruction format

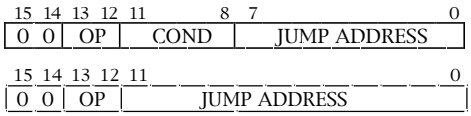


New literal Instruction format

Control Transfer Instructions

Conditional and unconditional jumps with absolute addresses

Instruction format



- Extra points for making conditional branch PC relative
- Requires assembler changes

Instructions

OP_{bin}	operation	mnemonic
00	Jump False	`JF.cond JPC`
01	Jump True	`JT.cond JPC`
10	Uncond. Jump	`J JPC`

$COND_{bin}$	condition	mnemonic
0100	ALU result negative	`.NEG`
0101	ALU result zero	`.ZERO`
0110	ALU carry	`.CARRY`
0111	ALU result negative or zero	`.NEGZERO`
0000	TRUE	`.TRUE`
1000	External Condition	`.EXT`

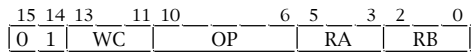
- Could add up to 8 external conditions

Condition codes are only set by ALU instructions

Memory Instructions

Register indirect addressing mode

Instruction format



Instructions

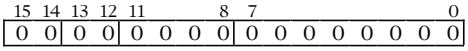
OP_{hex}	operation	mnemonic
08	C = Mem[A]	`LOAD C, A`
10	Mem[A] = B	`STORE A, B`

NOP Instruction

NOP instructions are useful in pipelined processors

Many different NOP instruction encodings are possible

NOP Jump False on condition TRUE: JF.TRUE 0x00



I/O Devices I

- VGA
  - Memory mapped I/O
  - Pick two addresses in processor address space
    - Addr1 = BRAM Address
    - Addr2 = BRAM data

```
STORE Addr1, X //Use X as BRAM location
LOAD  Addr2, A //Load from BRAM location
STORE Addr2, A //Store to BRAM location
```

- Use VGA
  - Manipulate a simple polygon on the display
    - Flash a square on and off
    - move a square from side to side
  - Design needs to run at least 25MHz

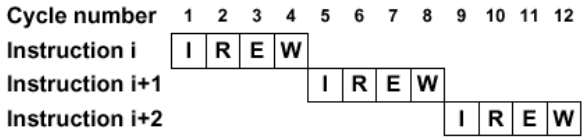
I/O devices II

- Connect free running counter to EXT
  - Choose 1 bit or multiple bits
  - Extra: use memory mapped I/O to reset
- Use EXT in timing loops

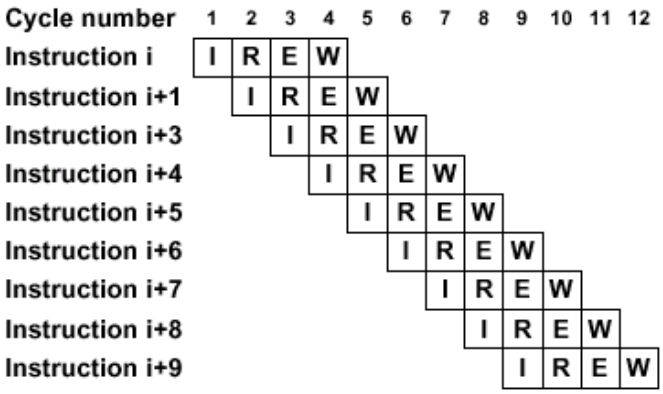
```
_LAB1 JF.EXT _LAB1 //spin waiting for posedge
     < Body of timing loop >
_LAB2 JT.EXT _LAB2 //spin waiting for negedge
      J      _LAB1
```

Instruction Execution Steps

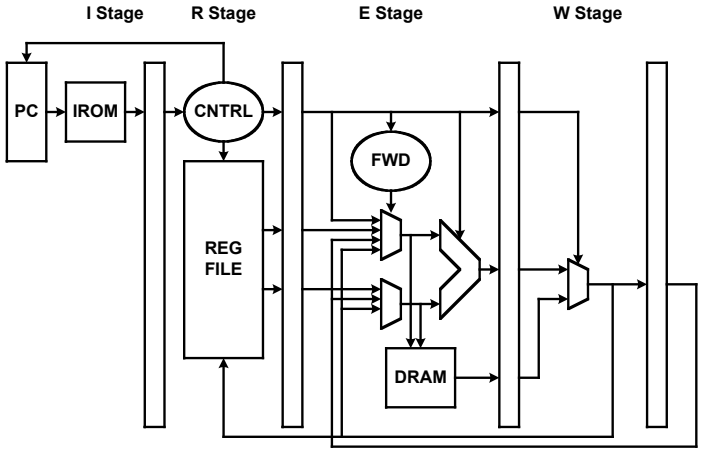
- 4 Step Sequence
  - Step I Fetch instruction from Instruction Memory
  - Step R Read operands from registers (A, B)
  - Step E Execute instruction, set condition codes
  - Step W Write results to register C
- One stage per step
- Each instruction goes through all four stages
  - Assume each stage takes one clock cycle



Pipeline



Pipelined CPU Block Diagram

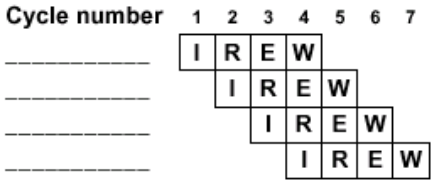


Bypassing/Forwarding

- Given the following code fragment

  ADD R1, R2, R3
  SUB R4, R1, R5
  XXX
  YYY

- What's going on in the pipeline?



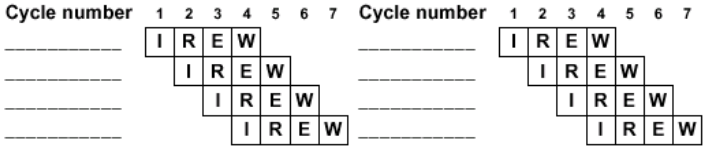
- How many different types of data hazards are there?

Control Transfer

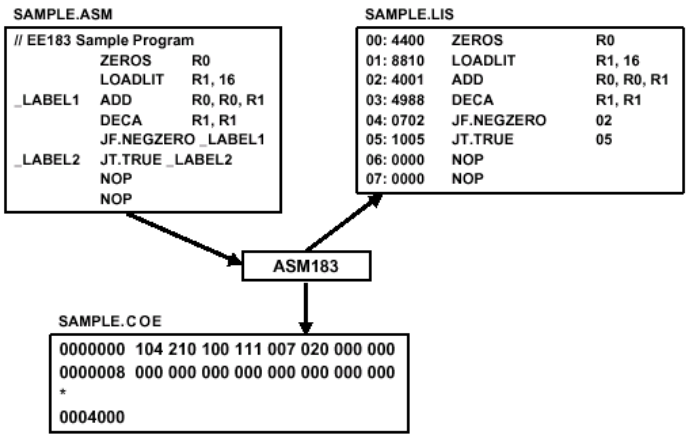
- Code fragment

```
00 ADD R1, R2, R3
01 JT.ZERO _taken
02 SUB R4, R5, R6
03 AND R7, R8, R1
…
_taken 11 NOR R7, R8, R1
```

- Branch Taken vs. Branch Not Taken



ASM183 (Assembler)



What do you get?

- Lab 3 Verilog
  - A lot of verilog given
  - Look through ALL of it
    - Some are not instantiated in the Lab 3 schematic
      - –e.g. **boolean.v**
- ASM183
  - Perl assembler
  - Perl Handout
    - How many already know perl?