

CSC 252: Processor Architecture

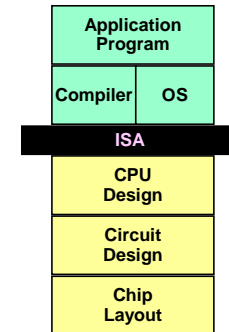
Guest Lecture
by
Sandhya Dwarkadas

2/28/2015

1

Instruction Set Architecture

- Assembly Language View
 - Processor state
 - Registers, memory, ...
 - Instructions
 - `addl, movl, leal, ...`
 - How instructions are encoded as bytes
- Layer of Abstraction
 - Above: how to program machine
 - Processor executes instructions in a sequence
 - Below: what needs to be built
 - Use variety of tricks to make it run fast
 - E.g., execute multiple instructions simultaneously



2/28/2015

2

Basic Issues in Instruction Set Design

- Goal:** find a language that makes it easy to build both the hardware and the compiler while maximizing performance and minimizing cost
- What operations (and how many) should be provided
 - How (and how many) operands are specified
 - What data types and sizes
 - How to encode these into consistent instruction formats

2/28/2015

3

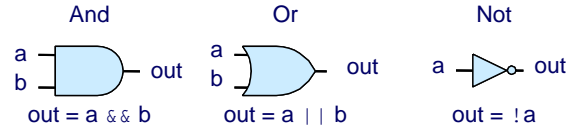
Overview of Logic Design

- Fundamental Hardware Requirements
 - Communication
 - How to get values from one place to another
 - Computation
 - Storage
- Bits are Our Friends
 - Everything expressed in terms of values 0 and 1
 - Communication
 - Low or high voltage on wire
 - Computation
 - Compute Boolean functions
 - Storage
 - Store bits of information

2/28/2015

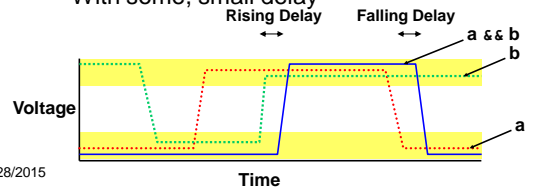
4

Computing with Logic Gates



- Outputs are Boolean functions of inputs
- Respond continuously to changes in inputs

- With some, small delay



2/28/2015

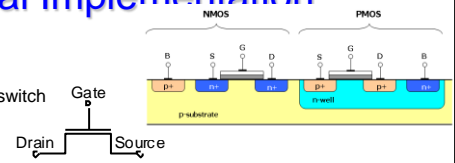
5

Physical Implementation

MOS Technology

Transistor

basic electrical switch



three terminal switch: gate, source, drain

voltage between gate and source exceeds threshold
switch is conducting or "closed"
electrons flow between source and drain

when voltage is removed,
the switch is "open" or non-conducting
connection between source and drain is broken

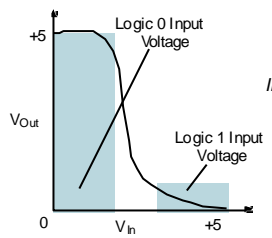
voltage-controlled switch

Building upon transistor switches, we can construct logic gates that implement various logic operations such as AND, OR, and NOT

Logic gates -- physical devices that operate over electrical voltages rather than symbols like 1 and 0.

Typical logic gate interprets voltages near Vdd as a logic 1 and those near 0V as a logic 0.

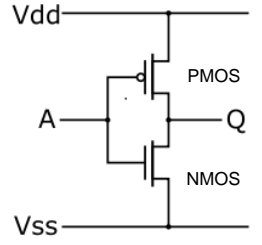
Circuit that implements logical negation (NOT)



1 at input yields 0 at output
0 at input yields 1 at output

Inverter behavior as a function of input voltage
input ramps from 0V to Vdd
output holds at Vdd for some range of small input voltages
then changes rapidly, but not instantaneously!

Inverter



2/28/2015

8

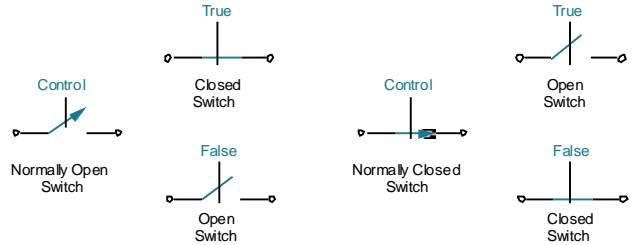
Digital Design Representation

1. Switches

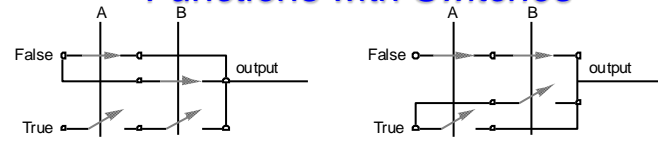
A switch connects two points (small circles) under control signal (arrow w/ line through it).

Normally Open when the control signal is 0 (false), the switch is open
when it is 1 (true), the switch is closed

Normally Closed when control is 1 (true), switch is open
when control is 0 (false), switch is closed



Implementation of AND and OR Functions with Switches



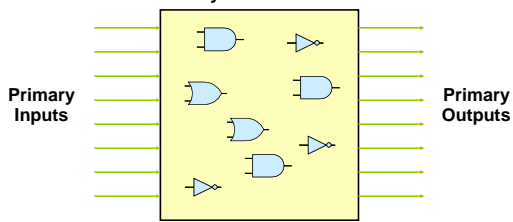
AND function
Series connection to TRUE

OR function
Parallel connection to TRUE

Series connection of Normally open -- AND
Parallel connection of Normally open -- OR

Combinational Circuits

Acyclic Network

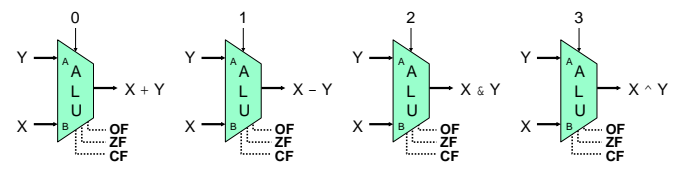


- Acyclic Network of Logic Gates
 - Continously responds to changes on primary inputs
 - Primary outputs become (after some delay) Boolean functions of primary inputs

2/28/2015

11

Arithmetic Logic Unit



- Combinational logic
 - Continuously responding to inputs
- Control signal selects function computed
 - Corresponding to 4 arithmetic/logical operations in Y86

2/28/2015

12

Sequential Logic: Memory and Control

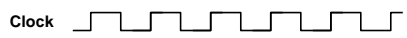
- Sequential:
 - Output depends on the **current** input values and the **previous** sequence of input values.
 - Are **Cyclic**:
 - Output of a gate feeds its input at some future time.
 - **Memory**:
 - Remember results of previous operations
 - Use them as inputs.
 - Example of use:
 - Build registers and memory units.

2/28/2015

13

Clocks

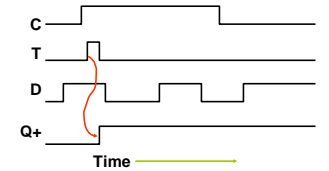
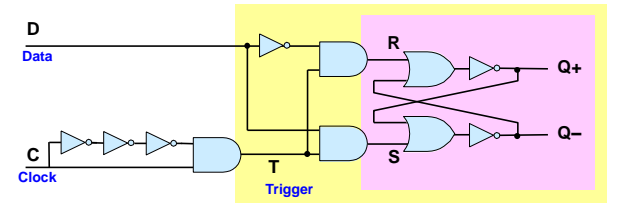
- Signal used to synchronize activity in a processor
- Every operation must be completed in the time between two clock pulses (or rising edges) --- the cycle time
- Maximum clock rate (frequency) determined by the slowest logic path in the circuit (the critical path)



2/28/2015

14

Edge-Triggered Latch

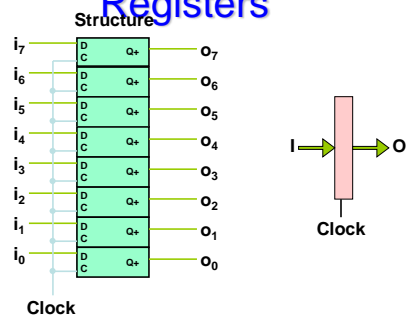


- Only in latching mode for brief period
 - Rising clock edge
- Value latched depends on data as clock rises
- Output remains stable at all other times

2/28/2015

15

Registers

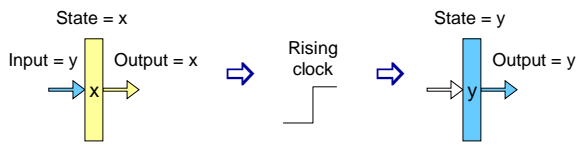


- Stores word of data
 - Different from *program registers* seen in assembly code
- Collection of edge-triggered latches
- Loads input on rising edge of clock

2/28/2015

16

Register Operation

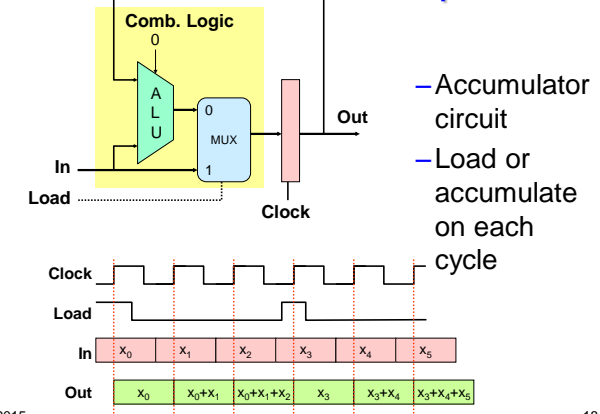


- Stores data bits
- For most of time acts as barrier between input and output
- As clock rises, loads input

2/28/2015

17

State Machine Example

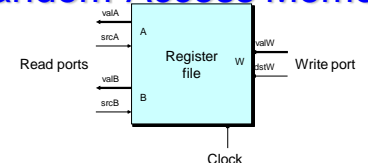


- Accumulator circuit
- Load or accumulate on each cycle

2/28/2015

18

Random-Access Memory

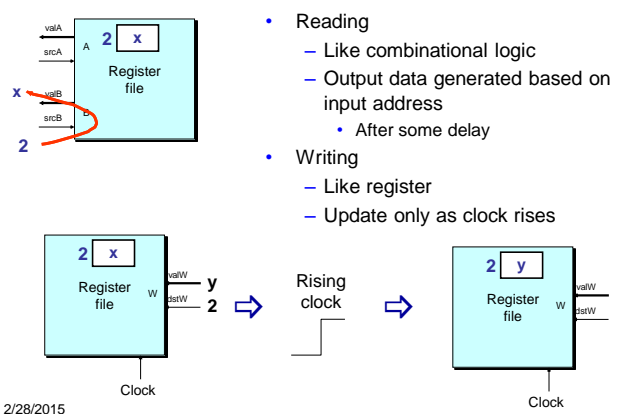


- Stores multiple words of memory
 - Address input specifies which word to read or write
- Register file
 - Holds values of program registers
 - %eax, %esp, etc.
 - Register identifier serves as address
 - ID 8 implies no read or write performed
- Multiple Ports
 - Can read and/or write multiple words in one cycle
 - Each has separate address and data input/output

2/28/2015

19

Register File Timing



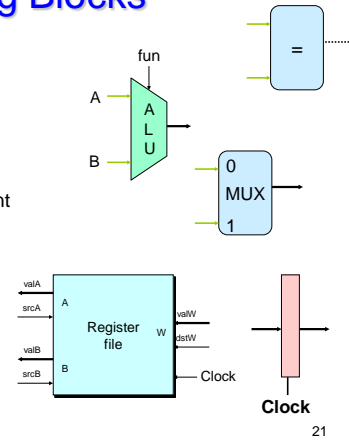
- Reading
 - Like combinational logic
 - Output data generated based on input address
 - After some delay
- Writing
 - Like register
 - Update only as clock rises

2/28/2015

20

Building Blocks

- Combinational Logic
 - Compute Boolean functions of inputs
 - Continuously respond to input changes
 - Operate on data and implement control
- Storage Elements
 - Store bits
 - Addressable memories
 - Non-addressable registers
 - Loaded only as clock rises

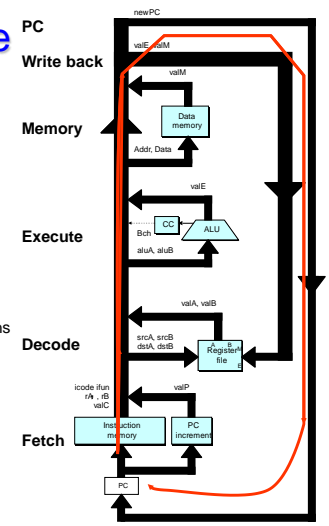


2/28/2015

21

SEQ Hardware Structure

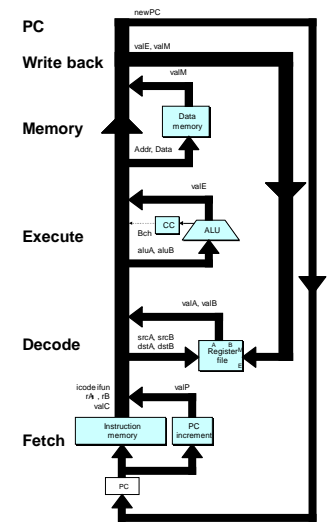
- State
 - Program counter register (PC)
 - Condition code register (CC)
 - Register File
 - Memories
 - Access same memory space
 - Data: for reading/writing program data
 - Instruction: for reading instructions
- Instruction Flow
 - Read instruction at address specified by PC
 - Process through stages
 - Update program counter



2/28/2015

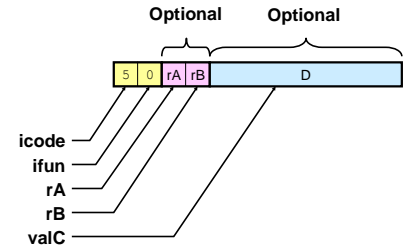
SEQ Stages

- Fetch
 - Read instruction from instruction memory
- Decode
 - Read program registers
- Execute
 - Compute value or address
- Memory
 - Read or write data
- Write Back
 - Write program registers
- PC
 - Update program counter



2/28/2015

Instruction Decoding

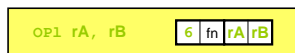


- Instruction Format
 - Instruction byte icode:ifun
 - Optional register byte rA:rB
 - Optional constant word valC

2/28/2015

24

Executing Arith./Logical Operation



- Fetch
 - Read 2 bytes
- Decode
 - Read operand registers
- Execute
 - Perform operation
 - Set condition codes
- Memory
 - Do nothing
- Write back
 - Update register
- PC Update
 - Increment PC by 2

2/28/2015

25

Possible Implementation Strategies

- Single-cycle control
- Multi-cycle control
- Pipelined (in-order execution)
- Superscalar and out-of-order execution

2/28/2015

26

Single-Cycle Control

- Cannot use any hardware unit twice in one cycle
 - Multiple datapath elements (e.g., ALU and PC incrementer) if needed for multiple purposes
 - Multiple reads or writes to memory or register file require multiplexing
- Clock cycle must accommodate instruction with the longest latency

2/28/2015

27

Multi-cycle Control

- Split single instruction into multiple pieces
- Execute individual pieces using hardwired finite state machine (FSM) or microprogramming
- Can finish simple instructions in fewer cycles
- Can run clock faster
- Still execute a single instruction at a time

2/28/2015

28

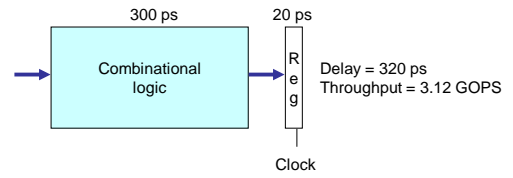
Pipelined Datapath

- Multiple instructions overlapped in execution
- Improve throughput, not individual instruction execution time
- Exploit parallelism among instructions in a sequential stream
- Balance length of each stage. Ideally -
 - $\text{Time between instrs}_{\text{pipelined}} = \text{Time between instrs}_{\text{non-pipelined}} / \text{Number of pipe stages}$

2/28/2015

29

Computational Example

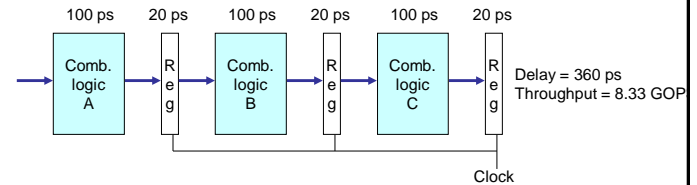


- System
 - Computation requires total of 300 picoseconds
 - Additional 20 picoseconds to save result in register
 - Must have clock cycle of at least 320 ps

2/28/2015

30

3-Way Pipelined Version

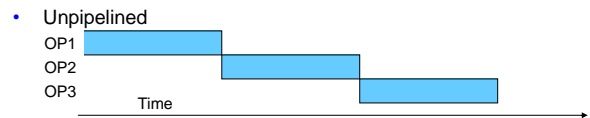


- System
 - Divide combinational logic into 3 blocks of 100 ps each
 - Can begin new operation as soon as previous one passes through stage A.
 - Begin new operation every 120 ps
 - Overall latency increases
 - 360 ps from start to finish

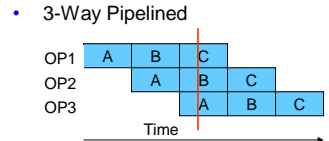
2/28/2015

31

Pipeline Diagrams



- Cannot start new operation until previous one completes

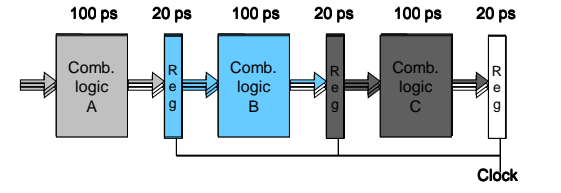
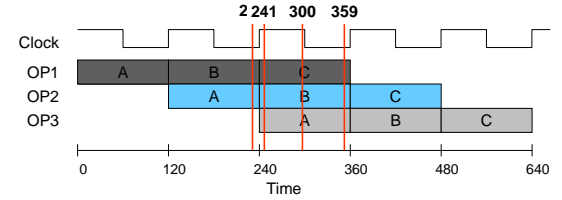


- Up to 3 operations in process simultaneously

2/28/2015

32

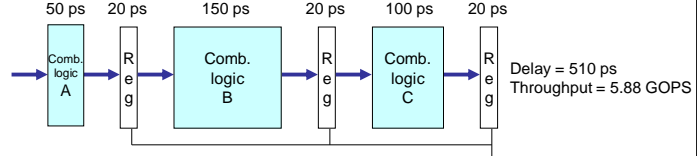
Operating a Pipeline



2/28/2015

33

Limitations: Nonuniform Delays

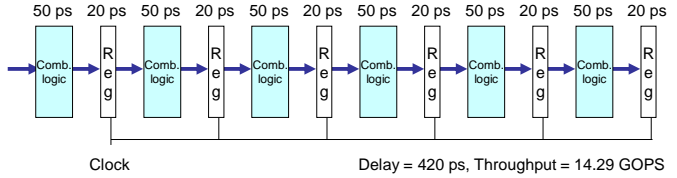


- Throughput limited by slowest stage
- Other stages sit idle for much of the time
- Challenging to partition system into balanced stages

2/28/2015

34

Limitations: Register Overhead



- As we try to deepen pipeline, overhead of loading registers becomes more significant
- Percentage of clock cycle spent loading register:
 - 1-stage pipeline: 6.25%
 - 3-stage pipeline: 16.67%
 - 6-stage pipeline: 28.57%
- High speeds of modern processor designs obtained through very deep pipelining

2/28/2015

35

Hazards

- Structural (e.g., instruction/data fetch)
- Control (e.g., jeq)
- Data (e.g., add followed by sub reading dst register of add)

2/28/2015

36

Data Dependencies

Clock

Time

- System
 - Each operation depends on result from preceding one

2/28/2015 37

Data Hazards

Clock

Time

- Result does not feed back around in time for next operation
- Pipelining has changed behavior of system

2/28/2015 38

Data Dependencies in Processors

```

1  irmovl $50, %eax
2  addl %eax, %ebx
3  mrmovl 100(%ebx), %edx
  
```

- Result from one instruction used as operand for another
 - Read-after-write (RAW) dependency
- Very common in actual programs
- Must make sure our pipeline handles these properly
 - Get correct results
 - Minimize performance impact

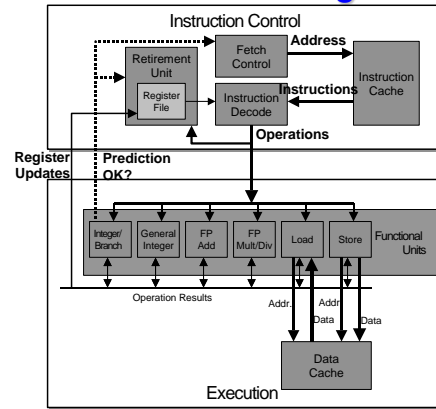
2/28/2015 39

Pipeline Summary

- Data Hazards
 - Most handled by forwarding
 - No performance penalty
 - Load/use hazard requires one cycle stall
- Control Hazards
 - Cancel instructions when detect mispredicted branch
 - Two clock cycles wasted
 - Stall fetch stage while `ret` passes through pipeline
 - Three clock cycles wasted
- Control Combinations
 - Must analyze carefully
 - Watch for subtle bugs that only arises with unusual instruction combination

2/28/2015 40

Modern CPU Design



2/28/2015

41

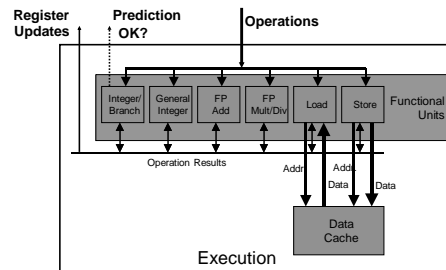
Instruction-level Parallelism

- Pipelining/super-pipelining
- Out-of-order execution
- Super-scalar
 - multiple instructions per pipeline stage
 - Dependences handled in hardware
- Very Large Instruction Word (VLIW)
 - Multiple instructions per pipeline stage
 - Dependences taken care of by compiler

2/28/2015

42

Execution Unit



- Multiple functional units
 - Each can operate independently
- Operations performed as soon as operands available
 - Not necessarily in program order
 - Within limits of functional units
- Control logic
 - Ensures behavior equivalent to sequential program execution

2/28/2015

43

Data Hazards: Forwarding

- The CDC 6600 Score-board architecture
- Tomasulo's generalized forwarding algorithm in the IBM360
 - Used reservation stations and shared buses

2/28/2015

44

Advanced Processor Design Techniques

- Trace caches
- Register renaming – eliminate WAW, WAR hazards
 - Register map table
 - Free list
 - Active list
- Speculative execution
- Value prediction
- Branch prediction
 - Branch history tables for prediction
 - Branch stack to save state prior to branch
 - Branch mask to determine instructions that must be squashed

2/28/2015

45

Processor Summary

- Design Technique
 - Create uniform framework for all instructions
 - Want to share hardware among instructions
 - Connect standard logic blocks with bits of control logic
- Operation
 - State held in memories and clocked registers
 - Computation done by combinational logic
 - Clocking of registers/memories sufficient to control overall behavior
- Enhancing Performance
 - Pipelining increases throughput and improves resource utilization
 - Must make sure maintains ISA behavior

2/28/2015

46

Disclaimer

- Parts of the slides were developed by and borrowed from slides the following textbook authors: Dave O'Hallaron and Randy Bryant, and Randy Katz. The slides are intended for the sole purpose of instruction of computer organization at the University of Rochester. All copyrighted materials belong to their original owner(s).

2/28/2015

47