

Today

- Arrays
 - One-dimensional
 - Multi-dimensional (nested)
 - Multi-level
- Structures
 - Allocation
 - Access
 - Alignment
- Unions
- Floating Point

1

Programming with SSE3

XMM Registers

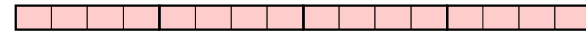
- 16 total, each 16 bytes
- 16 single-byte integers



- 8 16-bit integers



- 4 32-bit integers



- 4 single-precision floats



- 2 double-precision floats



- 1 single-precision float



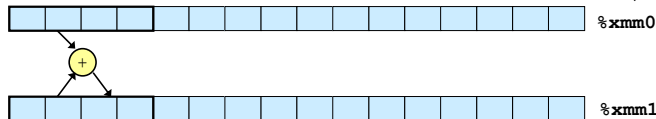
- 1 double-precision float



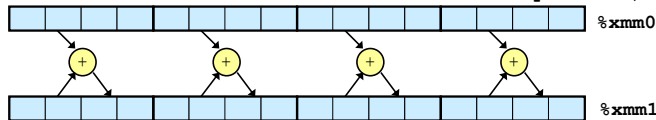
2

Scalar & SIMD Operations

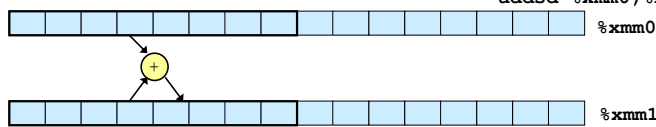
- Scalar Operations: Single Precision `addss %xmm0, %xmm1`



- SIMD Operations: Single Precision `addps %xmm0, %xmm1`



- Scalar Operations: Double Precision `addsd %xmm0, %xmm1`



3

FP Basics

- Arguments passed in `%xmm0, %xmm1, ...`
- Result returned in `%xmm0`
- All XMM registers caller-saved

```
float fadd(float x, float y)
{
    return x + y;
}
```

```
double dadd(double x, double y)
{
    return x + y;
}
```

```
# x in %xmm0, y in %xmm1
addss %xmm1, %xmm0
ret
```

```
# x in %xmm0, y in %xmm1
addsd %xmm1, %xmm0
ret
```

4

FP Memory Referencing

- Integer (and pointer) arguments passed in regular registers
- FP values passed in XMM registers
- Different mov instructions to move between XMM registers, and between memory and XMM registers

```
double dincr(double *p, double v)
{
    double x = *p;
    *p = x + v;
    return x;
}
```

```
# p in %rdi, v in %xmm0
movapd %xmm0, %xmm1 # Copy v
movsd (%rdi), %xmm0 # x = *p
addsd %xmm0, %xmm1 # t = x + v
movsd %xmm1, (%rdi) # *p = t
ret
```

5

Other Aspects of FP Code

- Lots of instructions
 - Different operations, different formats, ...
- Floating-point comparisons
 - Instructions `ucomiss` and `ucomisd`
 - Set condition codes CF, ZF, and PF
- Using constant values
 - Set XMM0 register to 0 with instruction `xorpd %xmm0, %xmm0`
 - Others loaded from memory

6

Breakout

Consider the following declaration of a two-dimensional array

```
int Array[n][n];
Assume n in %rdi;
      Array in %rsi;
      i in %rdx;
      j in %rcx
```

Write the assembly code (x86-based) to read `Array[i][j]` into register `%eax`

7

7

n X n Matrix Access

■ Array Elements

- Address $A + i*(C*K) + j*K$
- $C = n, K = 4$
- Must perform integer multiplication

```
/* Get element a[i][j] */
int var_ele(size_t n, int a[n][n], size_t i, size_t j)
{
    return a[i][j];
}
```

```
# n in %rdi, a in %rsi, i in %rdx, j in %rcx
imulq %rdx, %rdi # n*i
leaq (%rsi,%rdi,4), %rax # a + 4*n*i
movl (%rax,%rcx,4), %eax # a + 4*n*i + 4*j
ret
```

8

CSC 252: Processor Architecture

9

9

Instruction Set Architecture

- Assembly Language View
 - Processor state
 - Registers, memory, ...
 - Instructions
 - addl, movl, leal, ...
 - How instructions are encoded as bytes

How do we go from a sequence of instructions to actual execution?

Application Program	
Compiler	OS
ISA	
CPU Design	
Circuit Design	
Chip Layout	

10

10

Overview of Logic Design

- Fundamental Hardware Requirements
 - Communication
 - How to get values from one place to another
 - Computation – combinational logic
 - Storage – sequential logic
 - Clock to drive the next computation
- 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

11

11

Digital Signals

- Use voltage thresholds to extract discrete values from continuous signal
- Simplest version: 1-bit signal
 - Either high range (1) or low range (0)
 - With guard range between them
- Not strongly affected by noise or low quality circuit elements
 - Can make circuits simple, small, and fast

12

12

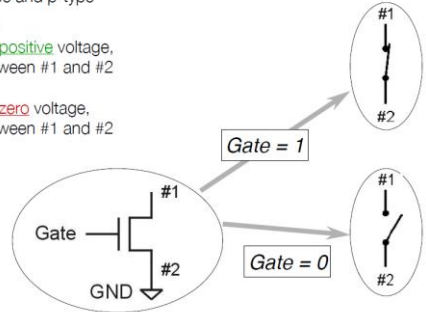
Basic Building Block: Transistors

MOS = Metal Oxide Semiconductor

- two types: n-type and p-type

n-type (NMOS)

- when Gate has **positive** voltage, short circuit between #1 and #2 (switch **closed**)
- when Gate has **zero** voltage, open circuit between #1 and #2 (switch **open**)



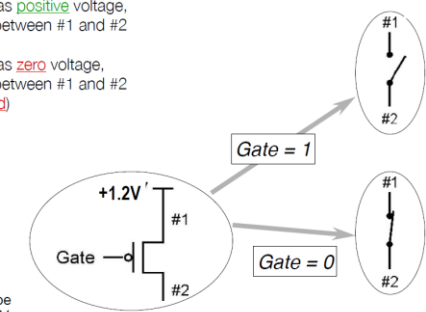
Terminal #2 must be connected to GND (0V).

13

Basic Building Block: Transistors

p-type is **complementary** to n-type (PMOS)

- when Gate has **positive** voltage, open circuit between #1 and #2 (switch **open**)
- when Gate has **zero** voltage, short circuit between #1 and #2 (switch **closed**)



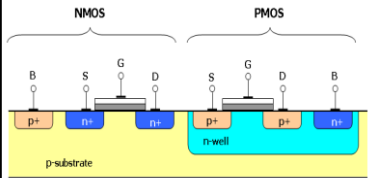
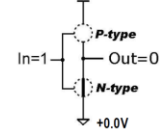
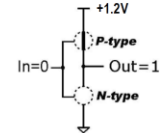
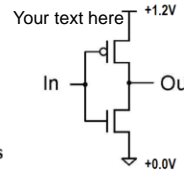
Terminal #1 must be connected to +1.2V

14

CMOS: Complementary MOS

- Use both n-type and p-type

Inverter (NOT Gate)

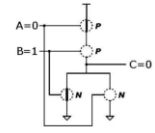
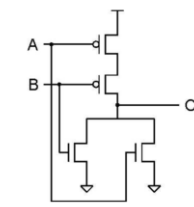


In	Out
0	1
1	0

15

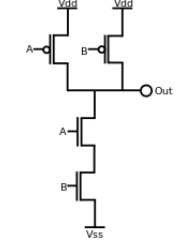
CMOS: NOR and NAND Gates

NOR Gate (NOT + OR)



A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

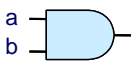
NAND Gate (NOT + AND)



16

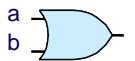
Computing with Logic Gates

And




out = a && b

Or



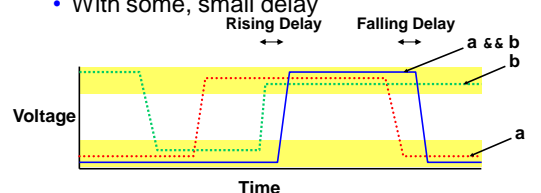
out = a || b

Not



out = !a

- Outputs are Boolean functions of inputs
- Respond continuously to changes in inputs
 - With some, small delay



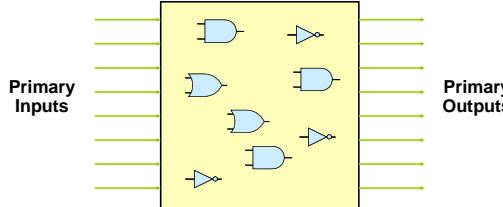
Time

17

17

Combinational Circuits

Acyclic Network



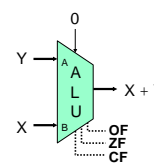
- Acyclic Network of Logic Gates
 - Continuously responds to changes on primary inputs
 - Primary outputs become (after some delay) Boolean functions of primary inputs

18

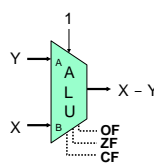
18

Arithmetic Logic Unit

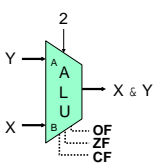
0



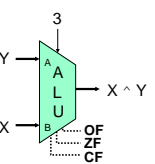
1



2



3



- Combinational logic
 - Continuously responding to inputs
- Control signal selects function computed
 - Corresponding to 4 arithmetic/logical operations in Y86
- Also computes values for condition codes

19

19

Sequential Logic: Memory and Control

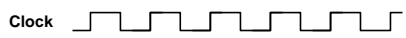
- Sequential:
 - Output depends on the **current** input values and the **previous** sequence of input values.
 - Are **Cyclic**:
 - Output of a gate feeds its input at some future time.
 - **Memory**:
 - Remember results of previous operations
 - Use them as inputs.
 - Example of use:
 - Build registers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

20

20

Clocks

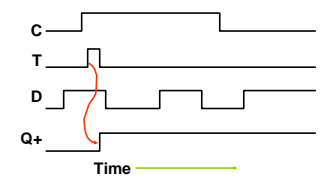
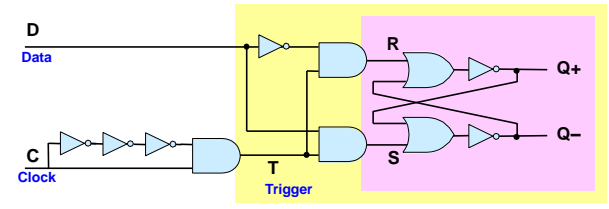
- Signal used to synchronize activity in a processor
- Every operation must be completed in the time between two clock pulses (or rising edges) --- the cycle time
- Maximum clock rate (frequency) determined by the slowest logic path in the circuit (the critical path)



21

21

Edge-Triggered Latch

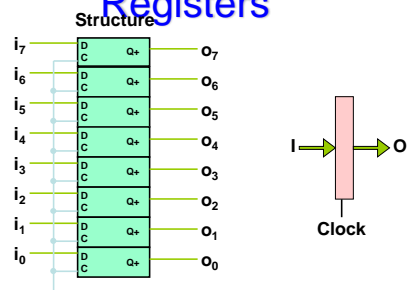


- Only in latching mode for brief period
 - Rising clock edge
- Value latched depends on data as clock rises
- Output remains stable at all other times

22

22

Registers

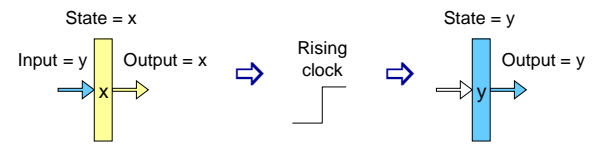


- Stores word of data
 - Different from *program registers* seen in assembly code
- Collection of edge-triggered latches
- Loads input on rising edge of clock

23

23

Register Operation



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

24

24

State Machine Example

– Accumulator circuit
– Load or accumulate on each cycle

25

25

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

26

26

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

27

27

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

28

28