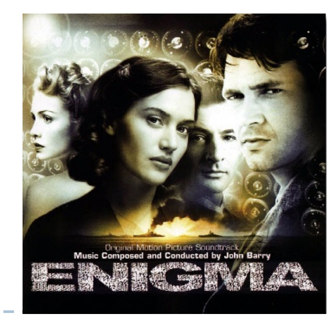
# Lecture 1 – History and Overview

CSE P567

## What is a Computer?

- Performs calculations
  - On numbers
  - But everything can be reduced to numbers
- Follows instructions (a program)
- Automatic (self-contained)
- Machine
  - But used to refer to people



## History of "Computers"

- People were hired to perform repetitious calculations
  - e.g. for making books of tables
- e.g. Gauss's human computer
  - Johan Dase
  - Hired to compute pi and factor integers

## Jacquard Loom

- Cards with holes are the instructions
- The holes control the hooks attached to warp threads
- First machine to use punch cards to control sequencing operation of a machine
- But not a calculator

courtesy Wikipedia

## Charles Babbage

- Difference engine #2 (1849)
  - Compute 7-th order polynomials to 31 decimal places
  - Mechanically – without mistakes
  - Faster than humans
- Method of differences
  - e.g `f(x) = x² - 2x + 4`

| x | f(x) | 1st difference | 2nd difference |
|---|---|---|---|
| 1 | 3 | | |
| 2 | 4 | 1 | |
| 3 | 7 | 3 | 2 |
| 4 | 12 | 5 | 2 |

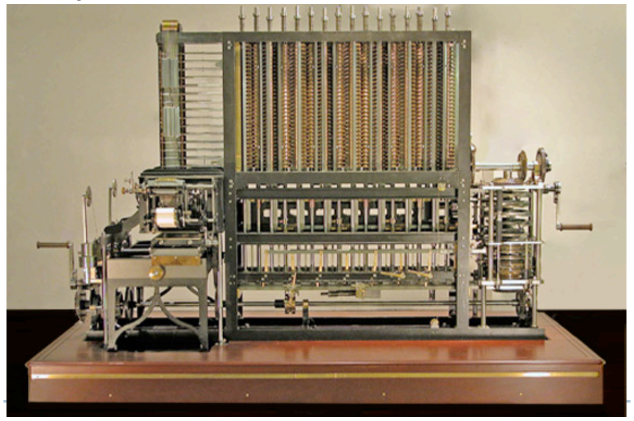
## Charles Babbage

- Difference engine #2 (1849)
  - Compute 7-th order polynomials to 31 decimal places
  - Mechanically – without mistakes
  - Faster than humans
- Method of differences
  - e.g `f(x) = x² - 2x + 4`

| x | f(x) | 1st difference | 2nd difference |
|---|---|---|---|
| 1 | 3 | | |
| 2 | 4 | 1 | |
| 3 | 7 | 3 | 2 |
| 4 | 12 | 5 | 2 |
| 5 | 19 | 7 | 2 |
| 6 | 38 | 9 | 2 |
| 7 | 39 | 11 | 2 |

## Difference Engine

- 1800's technology not good enough
- Replica recently completed and on display at the Computer Museum

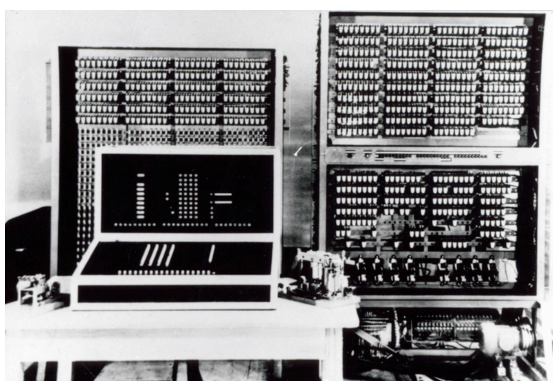


Difference Engine Video

courtesy Computer History Museum

## 1941: Z3 Computer – KonradZuse

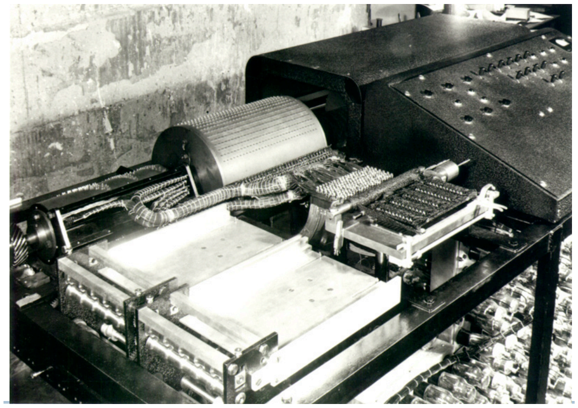
- 2300 relays
- Floating-point binary arithmetic



courtesy Computer History Museum

## 1942: Atanasoff-Berry Computer

- Iowa State College
- Not fully functional, but won patent dispute



courtesy Computer History Museum

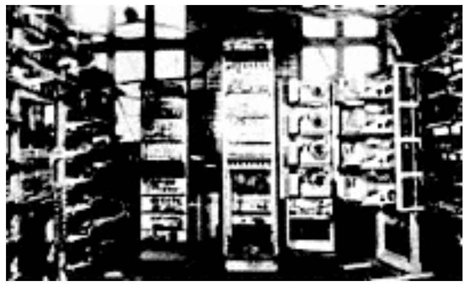
## 1946: ENIAC – Mauchly& Eckert

- Stored program computer
- Relays and switches
- .005 MIPS

courtesy Computer History Museum

## 1949: Manchester Mark 1

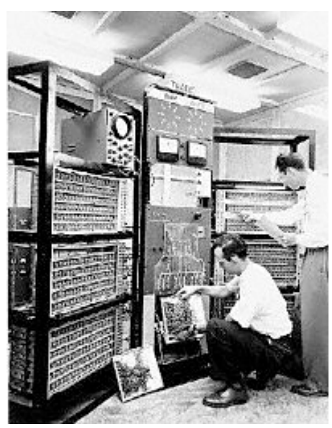
- Vacuum tube switches
- Memory: Cathode ray tube, magnetic drum
- addition delay – 1.8 microseconds



courtesy Computer History Museum

## 1955: Bell Labs TRADIC

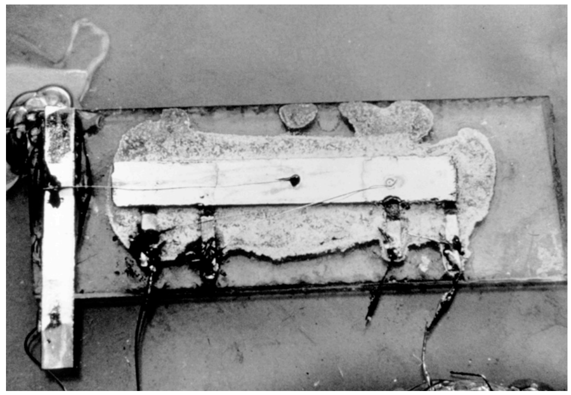
- First computer using transistors
- Reduced power by 20x



courtesy Computer History Museum

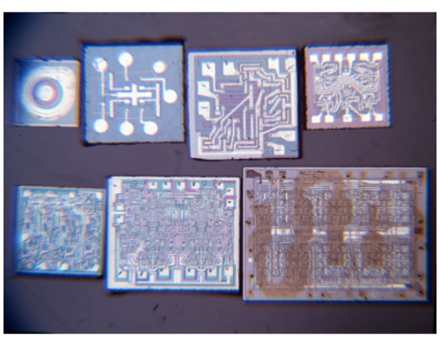
## 1958: First Integrated Circuit (Kilby)

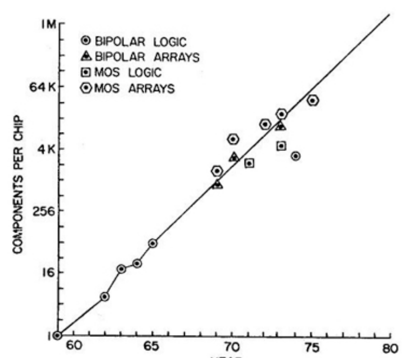
- 5 components on one sliver of germanium
  - Transistors, resistors, capacitors



courtesy Computer History Museum

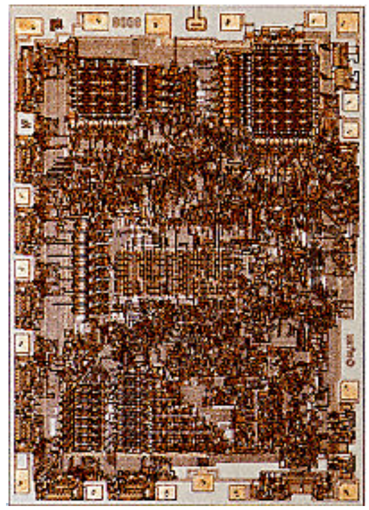
## 1965 - Moore's Law



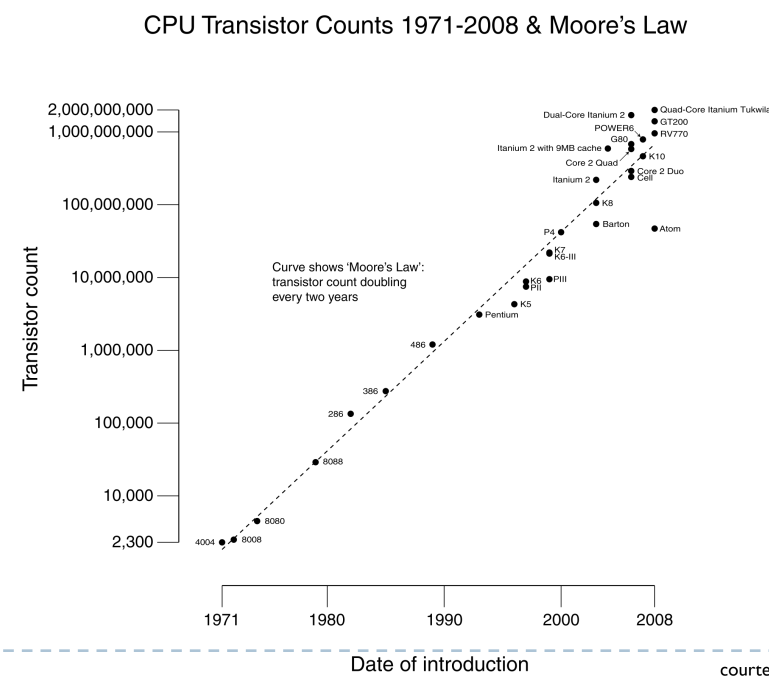


# 1971: First Microprocesor (Intel)

- 1971: 4004 – 4 bit processor
- 1972: 8008 – 8 bit processor



courtesy Computer History Museum




courtesy Wikipedia

## Hardware Design

- Ignoring scale, HW design reduces to:
  - Logic gates (AND, OR, INVERT)
  - Storage (registers)
- We can make these with ***switches***
- We can make switches with:
  - Relays
  - Vacuum tubes
  - Transistors (more later)
  - Nanotubes
  - ? ? ?

## Hardware Design

- "Register Transfer"
  - Move values from register to register
  - Perform some operation on these values
- CPU Example:
  - R1 = R2 + R3
  - Values already in R2 and R3
  - Move (connect) these values from R2 and R3 to the adder
  - Move (connect) the adder output to R1
  - Wait for clock to store new value in R1
    - Make sure only R1 is enabled

## Register Transfer

- CPU executes a sequence of instructions
  - Each is a register transfer
- Why can an instruction only do one thing?
  - Historically, ALUs and multipliers were expensive
  - Now we can supply many "function units"
- One instruction could specify multiple register transfers
  - They must be independent so they can execute in parallel
- All destination registers sample and hold simultaneously
  - Central clock
- Performance
  - How much happens before value is ready for latching?

## FIR Filter Example

- Mix of sequencing and computation

```
for (i = 0; i< N-T+1; i++)   {
y[i] = 0;
  for (j = 0; j< T; j++) {
y[i] += c[j] * x[i+j];
  }
}
```

- T adds and T multiplies for each `y[i]`
- Simple program uses at least 2T instructions
  - Plus loads and stores

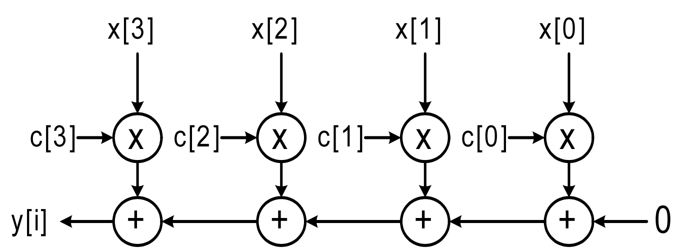
## FIR Filter Example

```
for (i = 0; i< N-T+1; i++)   {
y[i] = 0;
  for (j = 0; j< T; j++) {
y[i] += c[j] * x[i+j];
  }
}
```

```
r0 ← 0
ld  r2, C(r6)
r7 ← r5 + r6
ld  r3, X(r7)
r1 ← r2 * r3
r0 ←r0 + r1
etc.
```

## Direct Hardware Implementation

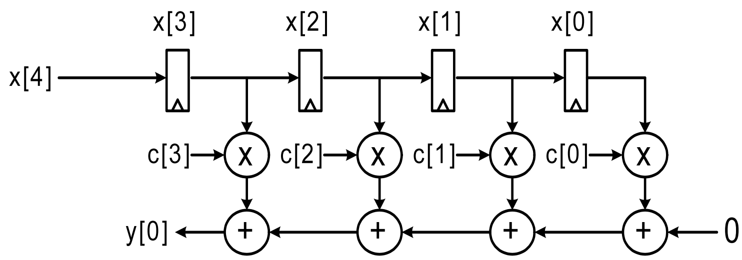
- If we can use as much hardware as we want:




- Convert time into space

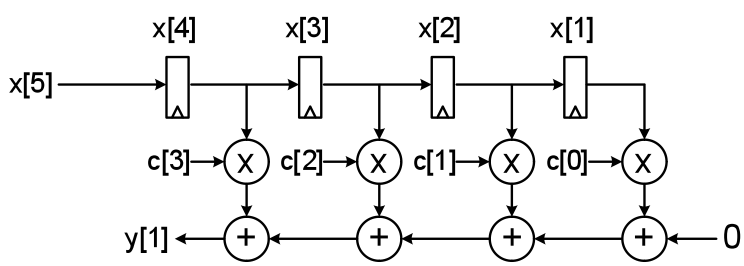
## Direct Hardware Implementation

- Reducing read bandwidth



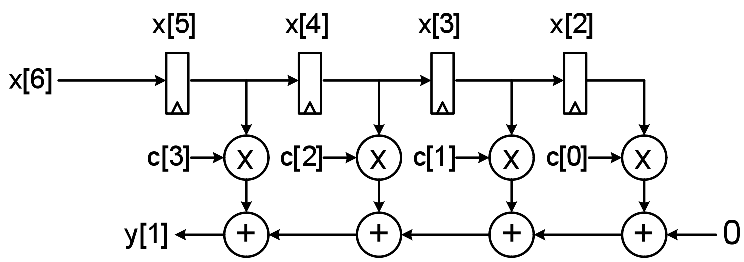

## Direct Hardware Implementation

- Reducing read bandwidth



## Direct Hardware Implementation

- Reducing read bandwidth




- Look at the longest register transfer…
  - Very slow clock
  - How can we make it faster?

## Register Transfer Summary

- We store values of interest in registers
- We compute on these values
  - And store the results in registers
- We can do multiple independent computations simultaneously
  - All results are clocked at the same time
- Example:
  - Shift register
  - Swap register values

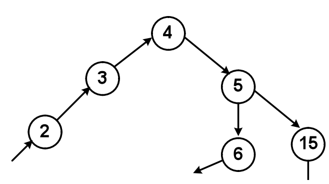
## Controllers

- Something must control what data transfers happen
  - Instruction execution
- Finite state machine
  - Inputs – status signals, e.g. result of comparison
  - Outputs – signals that select registers, enable registers
  - Set of states
  - Next state equation
  - Output equation

## Finite State Machines (FSMs)

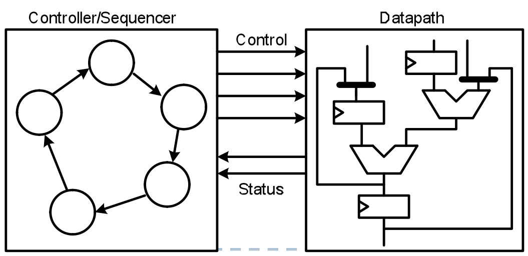
- Set of states (instruction addresses)
- Sequence through those states (next state equation)
  - State register has state (e.g. PC)
  - e.g. PC = PC + 1
  - Move from one state to the next on clock
  - May depend on input (conditional branch)
- Each state specifies instruction (output equation)
- Example

```
0:  r0 ← 0
1:  r1 ← r2 * r3
2:  r2 ←r1 * r1
3:  r0 ←r0 + r2
4:  cmp  r0, r4
5:  bge  . + 10
```



## Controller + Datapath

- Very common design methodology
- Controller specifies what to do in each clock cycle
  - Could be multiple, complicated things
- Datapath does it
  - Register transfer
- Note that controller uses register transfer as well
  - State register




## Designing Hardware

- What operations need to be done?
  - Provide function units
- What values are needed?
  - Provide registers
- In what order should the operation be executed?
  - Including parallelism
  - Design controller/sequencer (FSM)
- Then we need to connect everything together

## Hardware Systems

- Multiple, interacting hardware components
  - Multiple controller & datapaths
  - Memories
  - Disk controllers
  - Network interfaces
  - Physical interfaces (lights, motors, sensors, etc.)
  - etc.
- Connected together using interfaces and communication buses

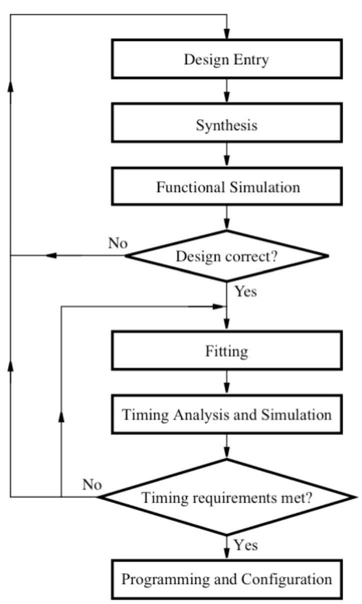
## Communication Buses

- Point-to-point
- Single master/multiple slave
- Multiple master
- Synchronous vs. Asynchronous
- Parallel vs. Serial
- Speed constrained by electrical considerations
  - Impedencemis-match
  - Ringing and reflections
  - Crosstalk
  - Return paths
  - Single-ended vs. differential
  - Inductive effects (di/dt)

## Implementation Alternatives

- Custom IC
  - Design mostly by hand – expensive
    - Intel and a few others
  - Send to foundry for fabrication – expensive and slow
- ASIC (semi-custom)
  - Rely on design tools to generate circuits
    - Less efficient – much less expensive/time-consuming
  - Send to foundry for fabrication – expensive and slow
- FPGA
  - Relay on design tools to generate circuits
  - User "programs" circuit into the FPGA – no NRE
    - Cheap and fast
  - Circuits are slower and bigger (no free lunch)

## Design Methodology




HDL (Verilog), schematics

Altera Quartus II

Mentor ModelSim

Altera Place and Route (Quartus)

AlteraQuartus STA (no simulation)

Altera Qartus

## Design Methodology

- Same flow for ASICs and FPGAs
  - Only details are different
- We will focus on using HDLs
  - Virtually all design is done with HDLs
- Verilog vs. VHDL
  - A matter of taste – they are more-or-less equivalent
  - Verilog – simple syntax, easy to learn
  - VHDL – more verbose, support for complex systems
  - We will use Verilog

## Verilog

- Syntax is reminiscent of C (or Java)
- Semantics is NOT!
- All blocks execute in parallel
- Register Transfer model
  - clock ticks: all registers latch new values (if enabled)
  - all logic computes new results with new register values
  - clock ticks: all registers latch new values (if enabled)
  - all logic computes new results with new register values
  - etc.

## A Word About the Lab

- We will give you a complete design in Verilog
  - Camera to LCD pipeline
- Lab 1 – Compile, download into hardware and test
  - Apply a small tweak to the design
- Lab 2 – Simple Verilog design and simulation
- Lab 3 – Implement adaptive threshold filter
- Lab 4 – Implement picture-in-picture
- Lab 5 – Chip layout tutorial
- Labs 6:10 – Embedded Systems
  - Rate-matching project

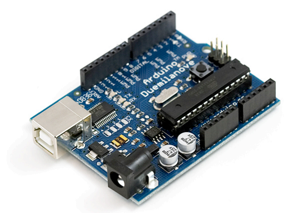
*Subject to change*

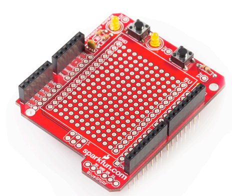
## Course Hardware

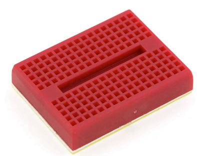
- Hard-hardware: Altera FPGA board
  - with camera and LCD screen
  - installed in 003 HW lab
  - run design tools at home (Windows)
- Soft-hardware: Arduino Atmel platform
  - very cool, extensible system
  - you buy in lieu of a textbook (~ $50)
  - run tools and hardware at home (Window or Mac)
  - we will supply widgets
    - LEDs, motors, accelerometers, light sensors

## Arduino Platform Details

- Arduino USB board - $29.95
  http://www.sparkfun.com/commerce/product_info.php?products_id=666
  ArduinoProtoShield Kit - $16.95
  http://www.sparkfun.com/commerce/product_info.php?products_id=7914
  Arduino Breadboard Mini Self-Adhesive - $3.95
  http://www.sparkfun.com/commerce/product_info.php?products_id=8800
  Total cost: $50.85 + shipping







Jan 7 is Free Day

## Labs

- Lab time is very limited!
  - We ask you to do much of the design at home
  - Come prepared to test and debug the design
  - Lab will be open before class so you can start early
- All tools are available for you to run at home
  - And in the lab of course