CmpSci 535

Computer Architecture

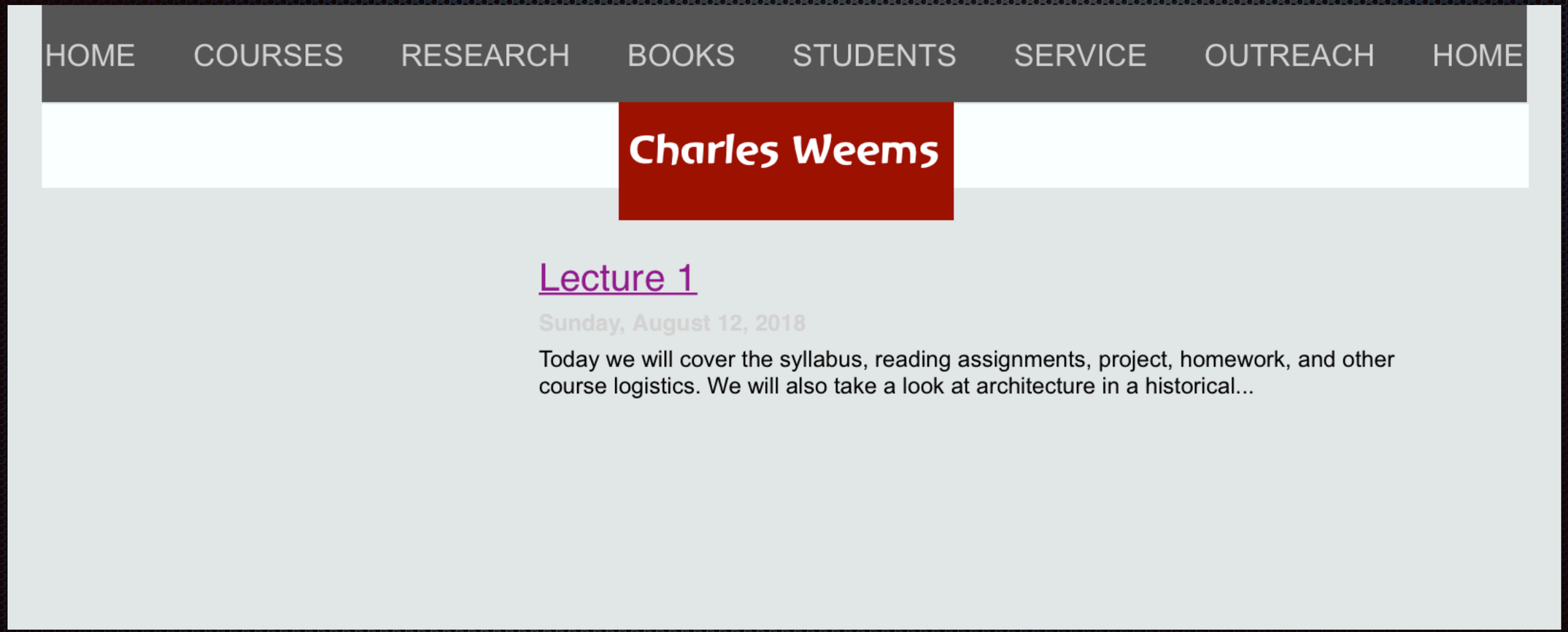
Intro

- Chip Weems
- weems@cs.umass.edu
- https://people.cs.umass.edu/~weems/homepage/index.html
- https://people.cs.umass.edu/~weems/homepage/courses/index.html
- Office: CS 342
- Office hours: Monday after class until 11:30, drop-ins, appointments

Web Page

https://people.cs.umass.edu/~weems/homepage/courses/index.html

Links to class notes



Note page contents

- Slides in pdf form
- Brief description of what was covered
- Any handouts as pdfs
- Announcements
- I will try to get these up within a day, but may occasionally fall behind due to other work, etc.

535 Course Goals

- See how computers really work
- Learn what factors affect their progress
- Design and simulate one to better understand tradeoffs and operation
- Compare some common architectures
- Explore acceleration mechanisms

Grading

- 18% Reading Homework
- 10% Class work/participation
- 4% Exam Prep Homework
- 10% Midterm
- 10% Final
- 15% Team Project Phase 1
- 14% Team Project Phase 2
- 14% Final Project Report
- 5% Team effort

Reading Homework

- For each reading on the schedule write 2+ questions
- Starts with February 3, ends April 29
- Due when indicated on schedule
- No credit if late unless prearranged
- Extra credit if early (10%) or extra questions (up to 5) or extra readings
- 18 readings, each 1%

Class Work/Participation

- There will be various in-class exercises
- Some will help start the project
- Others will solidify understanding of operation of architectural mechanisms
- Usually done in a team or group setting
- Can only be done in class (unless there is a prearranged absence)
- Number will be adjusted according to time available — 10% total

Exam Prep Homework

- Do on your own, to assess preparedness for exam
- Will be very similar to exam questions
- Can’t cover everything on exam (has to be written over a week before)
- Get feedback prior to exam to help with studying
- Each is 2%

Exams

- Open book, open notes
- Bring a calculator
- Will mimic homework questions
- Some questions may be on the project
- Midterm and final, 10% each
- Final only covers material after midterm

Team Project

- Create an instruction set architecture (ISA) and implement a simulation of it
    - The assembly language view of the machine
- Phase 1: Design the ISA, Implement a memory subsystem, Implement a simulator
    - Simulator has cache, pipeline, timing, basic UI, minimal subset of instructions
- Phase 2: Complete implementation of instructions and UI, with an assembler. Evaluate performance on at least two benchmarks
- Initial design proposal, series of demos, a final report
- Handout has details — also on course web page

Instruction Set Architecture (ISA)

- The raw assembly-language view of the machine, with no libraries or OS calls
- Designing one is an experience engineering tradeoffs
- Support basic ops on integers, memory access, control flow
- See course web site for a writeup

Project Grading Theme

- Meeting the minimal requirements is a B
- Falling short, getting behind schedule is less
- Taking initiative to go beyond requirements will earn a higher grade
- Many opportunities for extra credit

ISA Extra Credit

- Unusual features (special purpose, low power, secure, etc.)
- Special instructions (useful extensions, purpose-oriented, etc.)
- I/O devices (graphics, simulated sound, controls, network, etc.)
- Check with me before going too far -- it still needs to be feasible for implementation

Develop a Simulator

- Software represents all state in the machine: registers, memory, status, mode...
- Loads a binary program from a file
- Interprets instructions in proper order, updating simulated state same as hardware
- Like a VM in many ways

First step: Memory and Cache

- Memory (RAM) is slow (100 cycle access)
- Cache holds recently used instructions and data in a small, fast memory
- General memory unit class can instantiate as any level of cache or RAM
  - Instantiate RAM with null pointer to lower level
  - Cache has pointer to RAM (or to another cache level)
  - Basically a large integer array, where the index is the address

Basic Cache

- Direct-mapped
- 4 words per line
- Write-through, no allocate
- Unified
- Single cycle access (forward any wait response from lower level)

Cache Extras

- Associative caches — 2 or 4 way
- Alternate policies — wire back allocate, FIFO/LRU replacement, etc.
- Line length can be set (should be same at all levels)
- Additional levels (easy with generic memory class)
- Split Instruction/Data at top level

Initial Simulator

- Has cache and at least 5-stage pipeline
- Displays count of clock cycles
- Basic UI to display internal state, selected area of memory
- Single step execution
- Load/save programs, reset state

More Initial Simulation

- Minimal instructions (load, store, ALU, conditional branch)
  - Enough to run a simple looping program
- Instructions in binary (do NOT use strings)
- Only one memory -- holds code and data
- Ability to select area of memory to view
- Modes to run with cache and/or pipeline disabled

Pipeline

- Instructions pass through fetch, decode, execute, memory, write-back stages
- Each stage holds a different instruction -- provides parallelism
- Instructions may have dependences that cause stalls
  - Branches can cause flushes
- Can be turned off
  - Each instruction is added to the pipe after previous one exits write-back

Pipeline Extras

- Longer pipe
- Forwarding
- Branch prediction
- Interrupt handling
- Superscalar (multiple pipes)

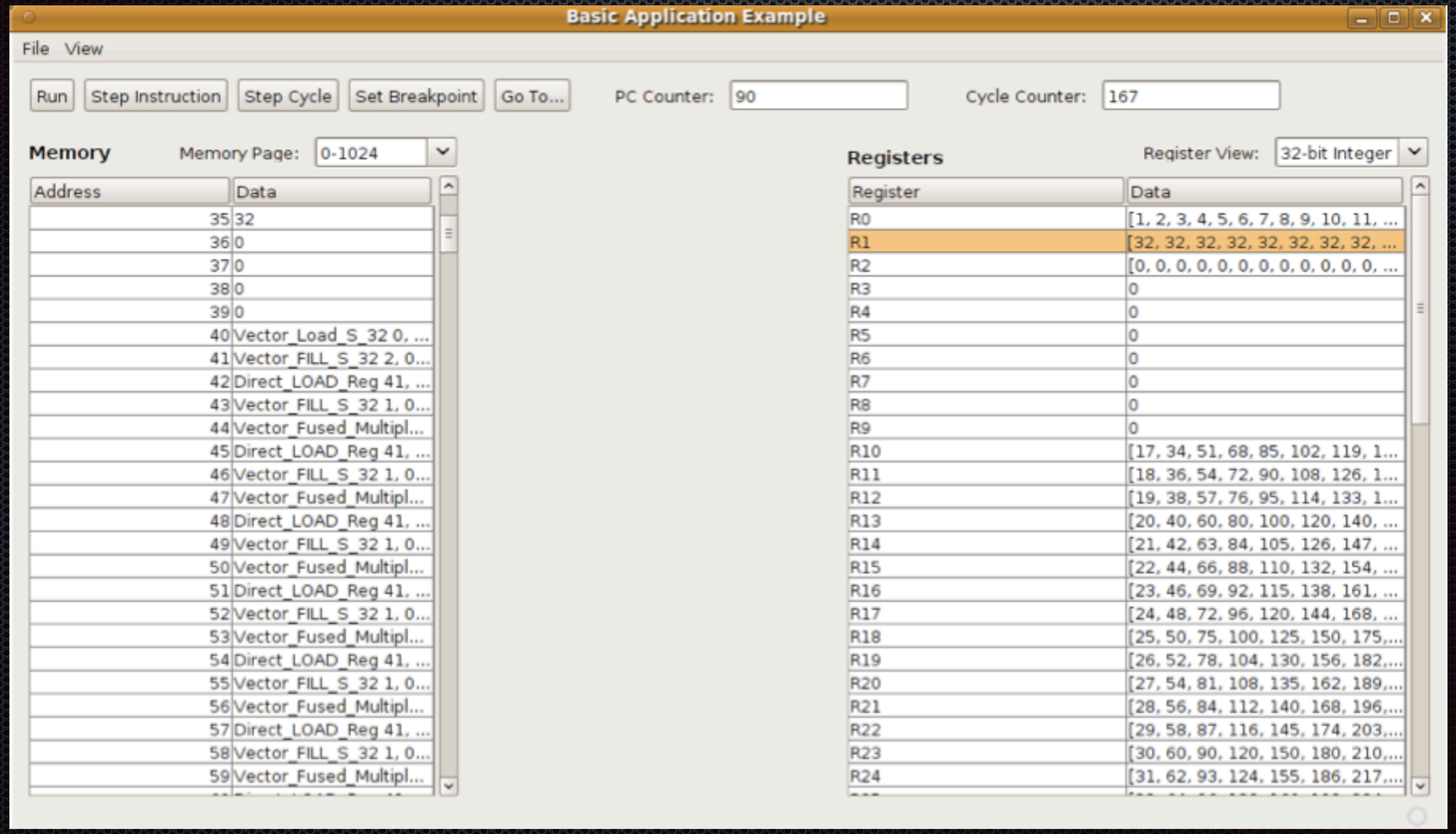
Project Strategy

- Cache and memory is easiest (comes first)
- Initial will take longer to implement - pipeline is challenging
- Just enough instructions to ensure cache and pipe are working
  - Then adding more instructions is step-by step process
- At same time, can build assembler, extend UI, to simplify development
- Last step is writing benchmarks on top of working system

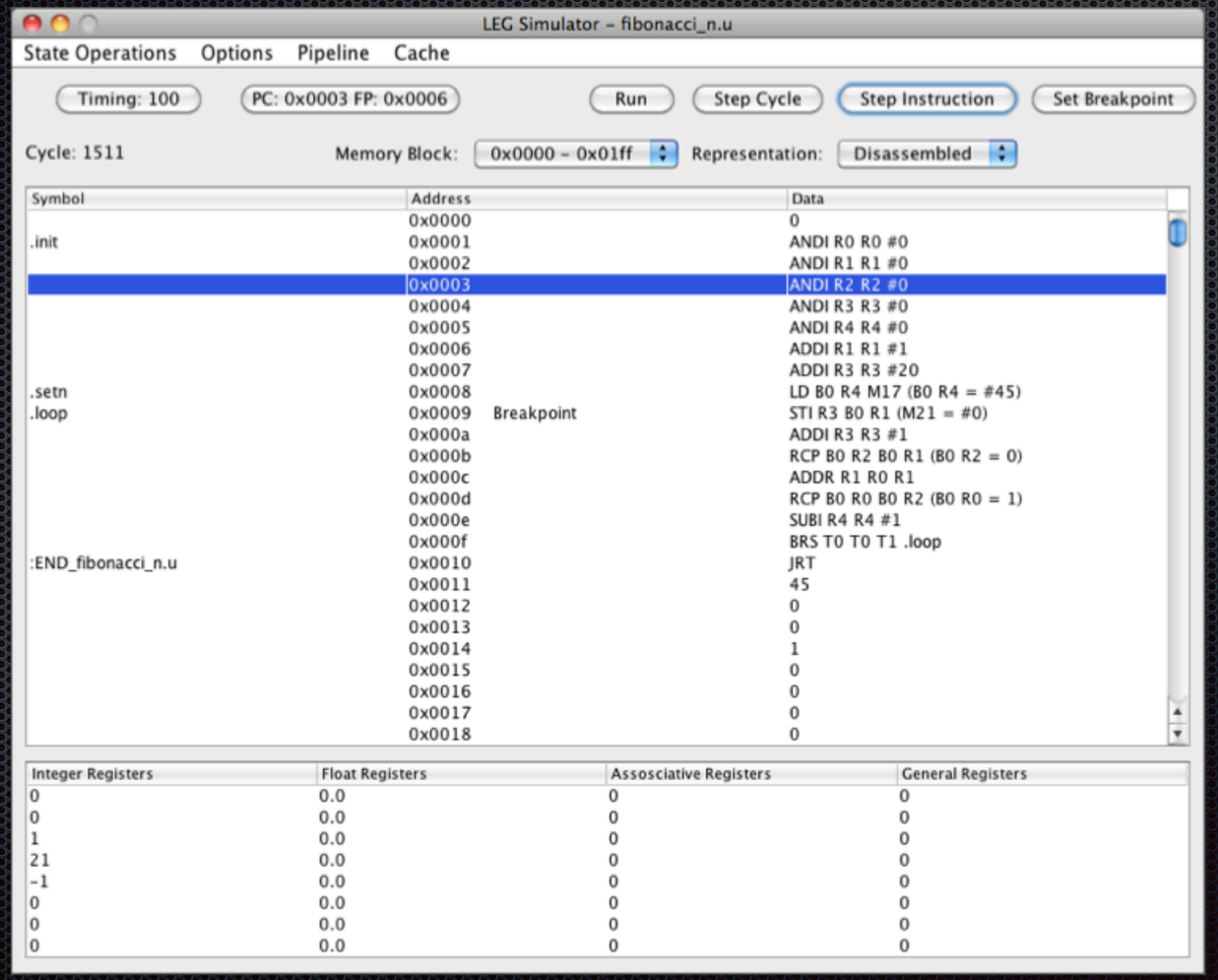
Full Simulator Extras

- Nicer, more flexible UI
- Macro assembler/mini-compiler
- More display modes (decimal, binary, floating point, string, instruction)
- Efficiency/speed tuning
- Getting ahead of schedule

Example UI


Basic Application Example
File View
Run
Step Instruction
Step Cycle
Set Breakpoint
Go To...
PC Counter: 90
Cycle Counter: 167
Memory
Memory Page: 0-1024
Address
Data
35 32
36 0
37 0
38 0
39 0
40 Vector_Load_S_32 0, ...
41 Vector_FILL_S_32 2, 0...
42 Direct_LOAD_Reg 41, ...
43 Vector_FILL_S_32 1, 0...
44 Vector_Fused_Multipl...
45 Direct_LOAD_Reg 41, ...
46 Vector_FILL_S_32 1, 0...
47 Vector_Fused_Multipl...
48 Direct_LOAD_Reg 41, ...
49 Vector_FILL_S_32 1, 0...
50 Vector_Fused_Multipl...
51 Direct_LOAD_Reg 41, ...
52 Vector_FILL_S_32 1, 0...
53 Vector_Fused_Multipl...
54 Direct_LOAD_Reg 41, ...
55 Vector_FILL_S_32 1, 0...
56 Vector_Fused_Multipl...
57 Direct_LOAD_Reg 41, ...
58 Vector_FILL_S_32 1, 0...
59 Vector_Fused_Multipl...
Registers
Register View: 32-bit Integer
Register
Data
R0 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...
R1 [32, 32, 32, 32, 32, 32, 32, 32, ...
R2 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
R3 0
R4 0
R5 0
R6 0
R7 0
R8 0
R9 0
R10 [17, 34, 51, 68, 85, 102, 119, 1...
R11 [18, 36, 54, 72, 90, 108, 126, 1...
R12 [19, 38, 57, 76, 95, 114, 133, 1...
R13 [20, 40, 60, 80, 100, 120, 140, ...
R14 [21, 42, 63, 84, 105, 126, 147, ...
R15 [22, 44, 66, 88, 110, 132, 154, ...
R16 [23, 46, 69, 92, 115, 138, 161, ...
R17 [24, 48, 72, 96, 120, 144, 168, ...
R18 [25, 50, 75, 100, 125, 150, 175,...
R19 [26, 52, 78, 104, 130, 156, 182,...
R20 [27, 54, 81, 108, 135, 162, 189,...
R21 [28, 56, 84, 112, 140, 168, 196,...
R22 [29, 58, 87, 116, 145, 174, 203,...
R23 [30, 60, 90, 120, 150, 180, 210,...
R24 [31, 62, 93, 124, 155, 186, 217,...

Another UI


LEG Simulator – fibonacci_n.u
State Operations
Options
Pipeline
Cache
Timing: 100
PC: 0x0003 FP: 0x0006
Run
Step Cycle
Step Instruction
Set Breakpoint
Cycle: 1511
Memory Block:
0x0000 – 0x01ff
Representation:
Disassembled
Symbol
Address
Data
0x0000 0
.init 0x0001 ANDI R0 R0 #0
0x0002 ANDI R1 R1 #0
0x0003 ANDI R2 R2 #0
0x0004 ANDI R3 R3 #0
0x0005 ANDI R4 R4 #0
0x0006 ADDI R1 R1 #1
0x0007 ADDI R3 R3 #20
.setn 0x0008 LD B0 R4 M17 (B0 R4 = #45)
.loop 0x0009 Breakpoint STI R3 B0 R1 (M21 = #0)
0x000a ADDI R3 R3 #1
0x000b RCP B0 R2 B0 R1 (B0 R2 = 0)
0x000c ADDR R1 R0 R1
0x000d RCP B0 R0 B0 R2 (B0 R0 = 1)
0x000e SUBI R4 R4 #1
0x000f BRS T0 T0 T1 .loop
:END_fibonacci_n.u 0x0010 JRT
0x0011 45
0x0012 0
0x0013 0
0x0014 1
0x0015 0
0x0016 0
0x0017 0
0x0018 0
Integer Registers
Float Registers
Assosciative Registers
General Registers
0 0.0 0 0
0 0.0 0 0
1 0.0 0 0
21 0.0 0 0
-1 0.0 0 0
0 0.0 0 0
0 0.0 0 0
0 0.0 0 0

Benchmarks

- Standard programs to evaluate performance
- At least integer exchange sort and matrix multiply (extra credit for more)
  - Data set must be big enough to more than fill cache and require at least 10 accesses per line
- Compare performance on all four modes of the simulator
  - No cache or pipe, cache only, pipe only, cache and pipe

Software Engineering

- Part of the project is to use good development methodology and tools
- Will be part of demos and reports
- Manage code in a repo, use a unit testing system, keep a punch list, use a wiki for documentation, etc.

Reports

- ISA Report -- complete description of the architecture, project management plan, team task assignments
- Final report -- amended ISA report plus simulator operation description, user manual, summary of software engineering methods applied, who did what, benchmark results analysis, what you have learned

Report Drafts

- First draft will be reviewed, given an interim grade with comments
- Final version can address comments, and its grade will replace the first one
- Lateness affects grade -- better to submit a partial draft and get feedback, since final version grade replaces draft

Report Due Dates

- First report draft: Wed., 2/12
- First report final: Wed., 2/24
- Final report draft: Wed., 4/22
- Final report final: Thu., 5/7 (at final exam)

Demos

- ISA presentation: Wed., 2/12 (brief overview)
- Memory/cache, timing demo: Mon., 3/2
- Initial Simulation: run simple looping program: Mon., 3/23
- Full ISA, UI, assembler: Wed., 4/8
- Benchmark execution: Wed., 4/22

Team Formation

- Next week, after add/drop settles
- You can form your team early
- Can also do project solo, but more work
- Joining a team is a commitment -- withdrawing will have a lasting impact

Team Size

- Optimal is two people
  - Three only if a very ambitious project with a strong management plan
- If you’re not sure that you’ll stay in the class, don’t form a team early
- Membership may change in early classes
- Project will start gradually during this time

First Team Meetings

- Exchange email, contact info
- Schedule regular team meeting time
- Plan collaboration strategy
- Talk about skill sets
- Decide on team organization

Read the Project Handout Carefully

- It answers the most common questions
- It has a lot of useful information/advice about the project
- It includes all of the demo dates and descriptions of what is expected
- You have access to it when you can’t reach me to ask questions

Survey

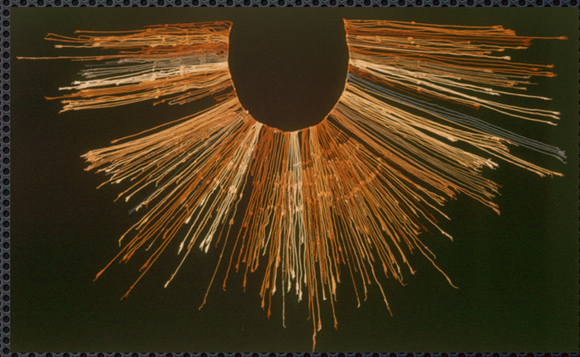
- Not a test -- just guidance for depth
- If you immediately know an answer, just write down enough to show that you do
- If you sort of remember it, then write something like “saw it before,” or “heard about it.”
- If it’s new to you, just put an X

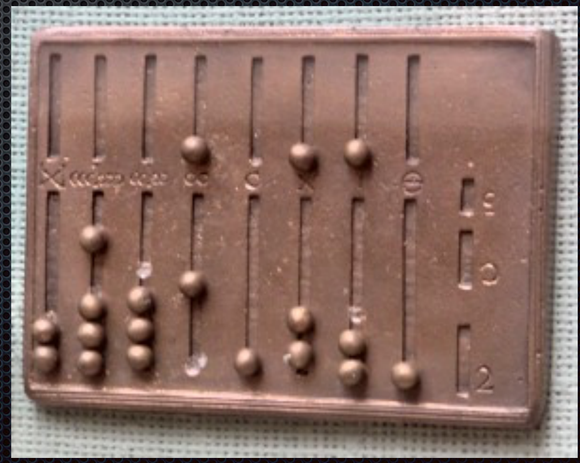
Historical Perspective

How we got here and, where exactly is here?

Early Computing

- 
- Others:
  - Abacus, knots, stones



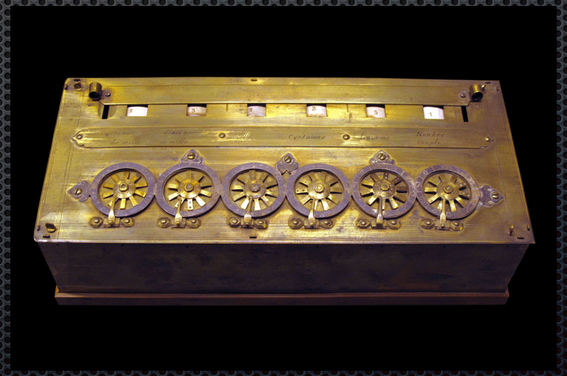


Functional Generations

- 1st: Memory aids -- increased accuracy, size of numbers
- 2nd: Automatic arithmetic -- greater accuracy, more complexity
- 3rd: Programmable -- extends accuracy to complex functions
- 4th: Reliable -- unlimited complexity, broader use, faster (to do more)
- 5th: Pervasive -- tolerate some failures

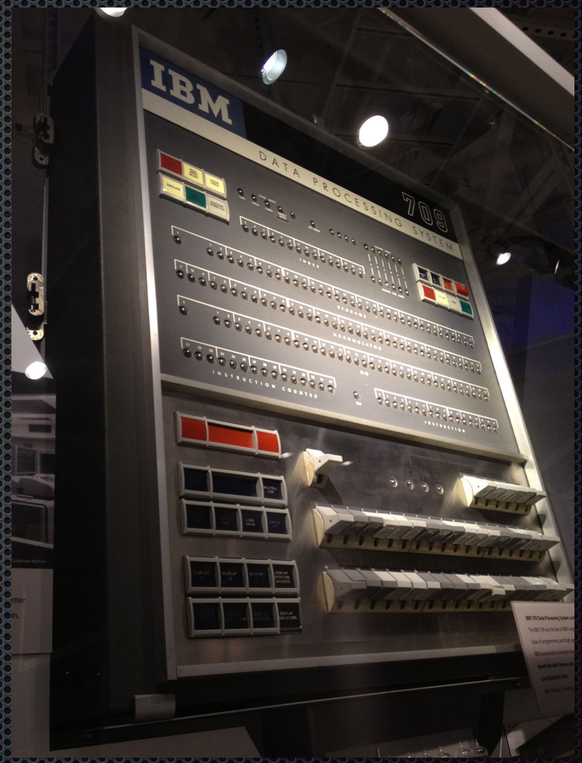
0th Tech Generation

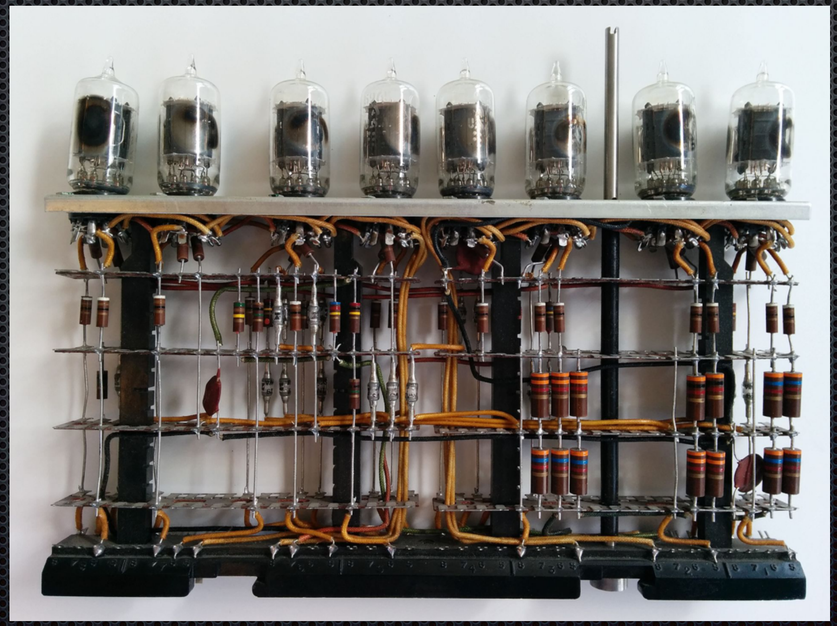
- What did the earliest computing aids do for us?
- What effect did the next group have?
- What was the original definition of “computer”



1st Tech Generation

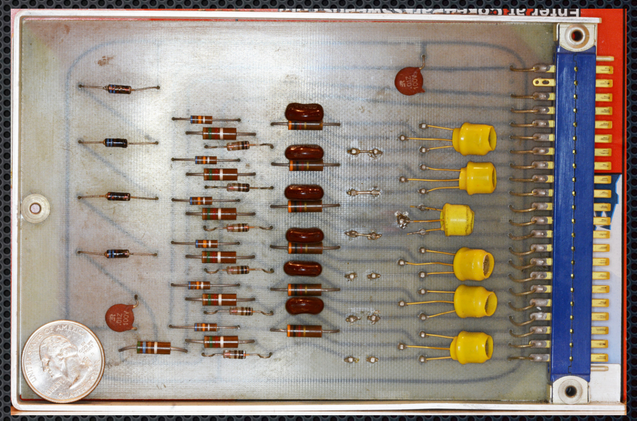
- What did electronic computers enable?
- What were the problems?

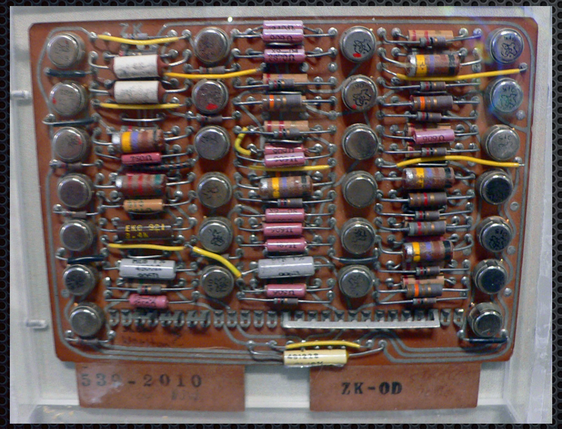




2nd Tech Generation

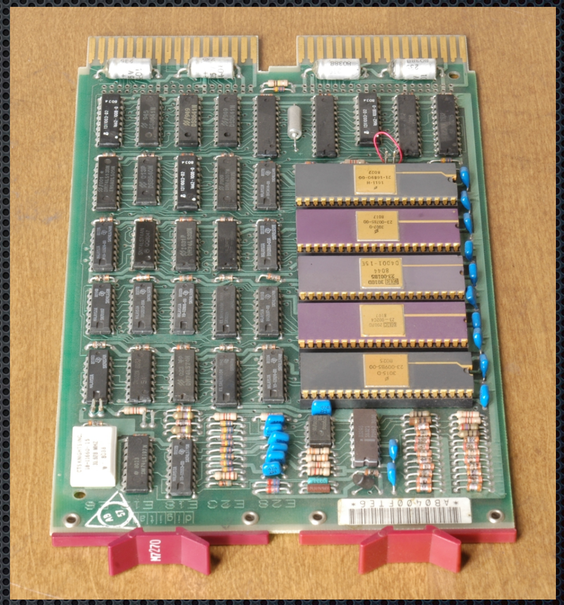
- Why switch to transistors?
- What are their advantages?

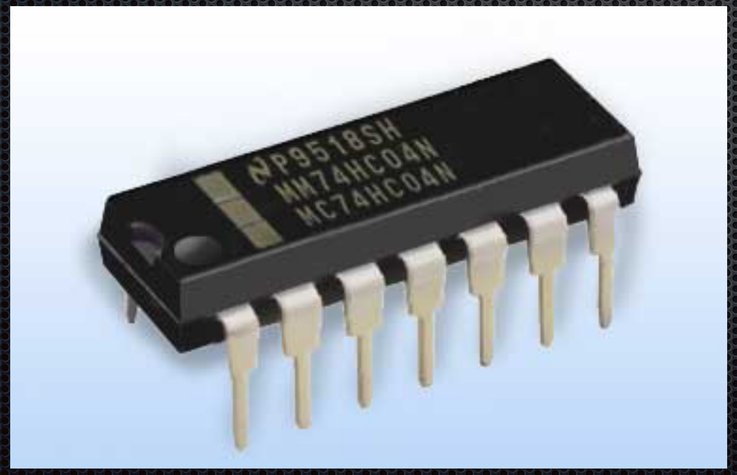




3rd Tech Generation

- Integrated circuits
- What are their main advantages?
- Note change in memory technology





4th Tech Generation

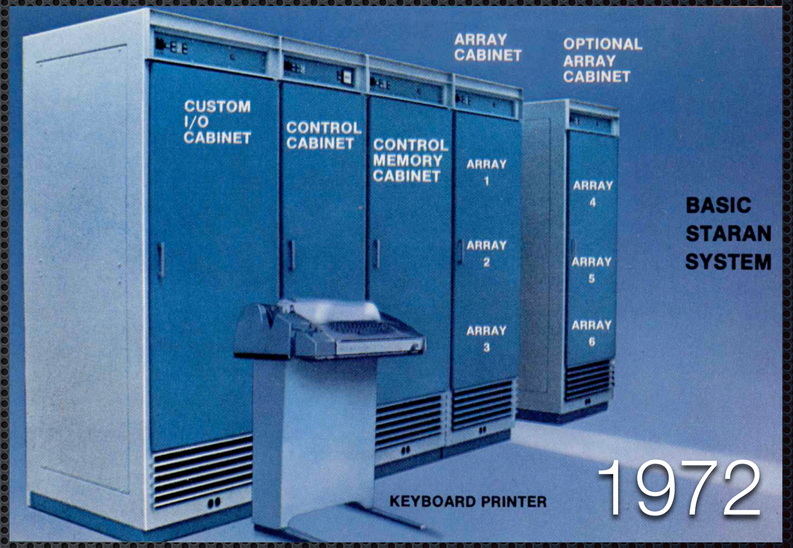
- Microprocessor
- Just more of the same -- why such a big deal?

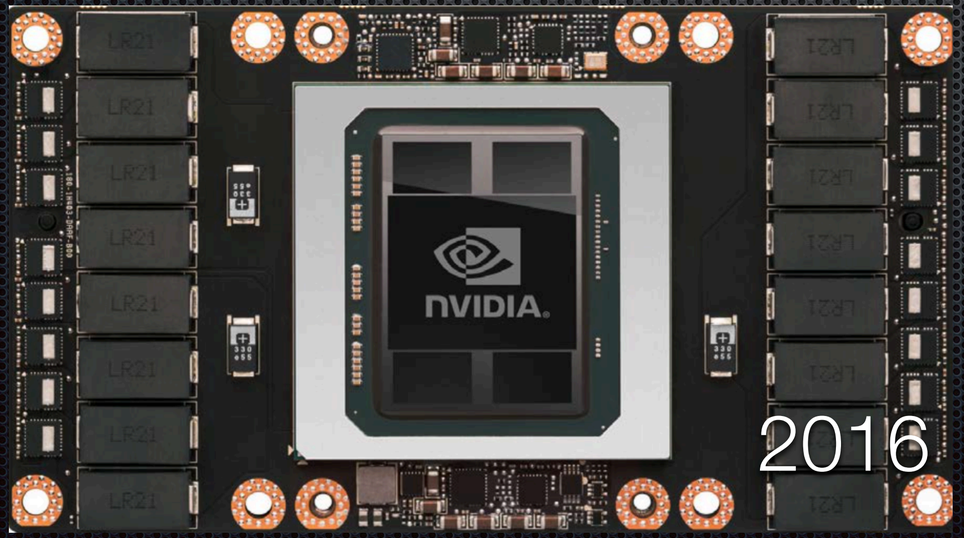
Microprocessors

- First were a huge step backward
- Grow in word size, add multiprocessing
- Then add pipeline parallelism (faster clock)
- Cache memory; one then multiple levels
- Superscalar (multiple pipelines)
- Multithreading

5th Tech Generation

- Parallel
- Conceived with first computers
- Why did it fail to take off?
- Why is it now taking over?



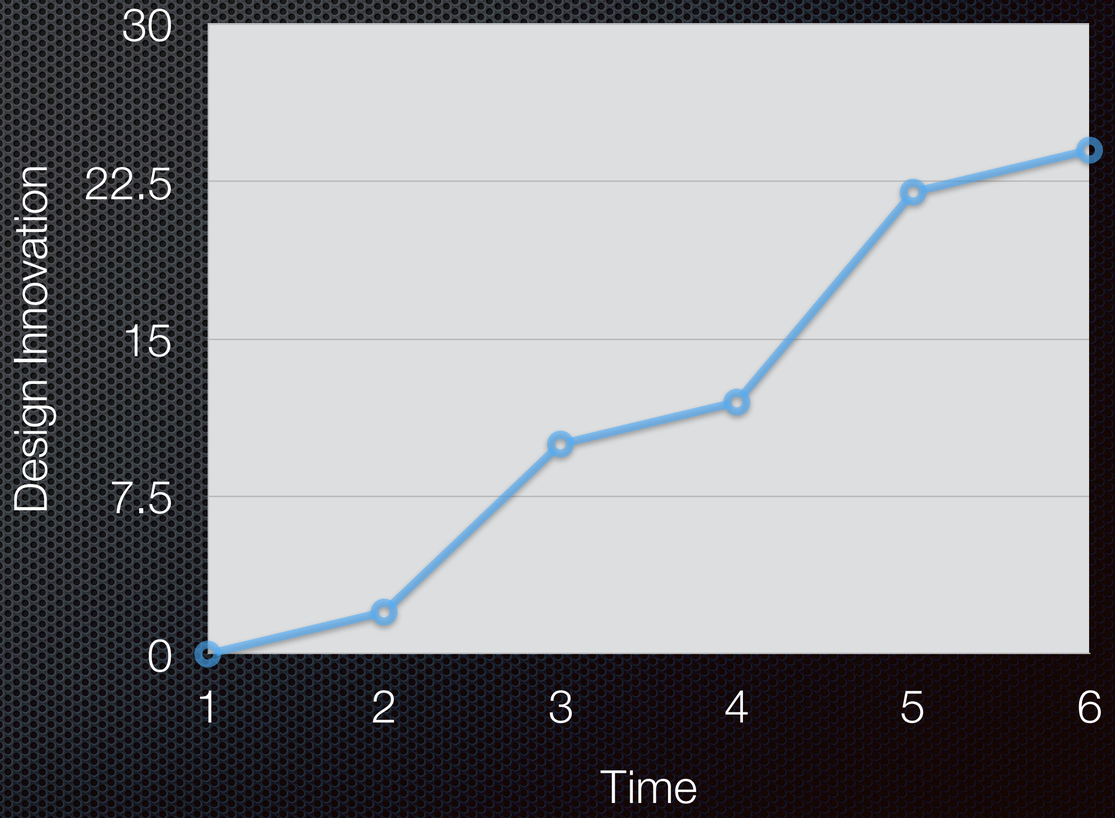


Hitting the Wall

- Faster clock needs more power
- Power becomes heat
- Heat slows circuits, increases power, wearout
- Greater complexity consumes more power for diminishing speedup
  - Longer relative distances for signals

Punctuated Equilibrium

- Gradual progress, then cross threshold
- Tend to reimplement earlier designs
- Then new designs appear
- Shakeout leads to stable, gradual progress



Parallelism

- Multiple cores don’t require common clock
  - Easy utilization of chip area
- Keep clock constant, but increase performance
- Flexible power management
- Immediately available parallel workloads

More Parallelism

- Run out of easy workload distribution
  - Harder to find work to drive 24-cores
- Shared memory becomes bottleneck
  - Hierarchy of sharing
- One size doesn’t fit all -- heterogeneity
- Challenges dominant programming model
  - Much more insidious bugs

The Future

- Low end eats profits, cuts R&D
- Potential for stagnation at high end
  - Likely to involve government support
- Greater diversity, heterogeneity, more embedded
- Technology shifts in memory construction - DRAM scaling ending
- New programming models: confusion precedes convergence