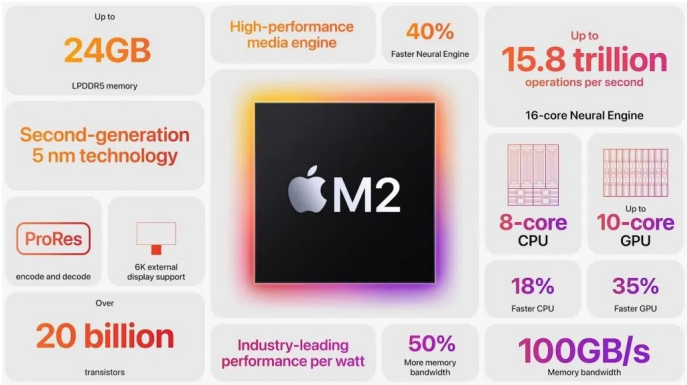
CS 152/252A Computer Architecture and Engineering

Sophia Shao

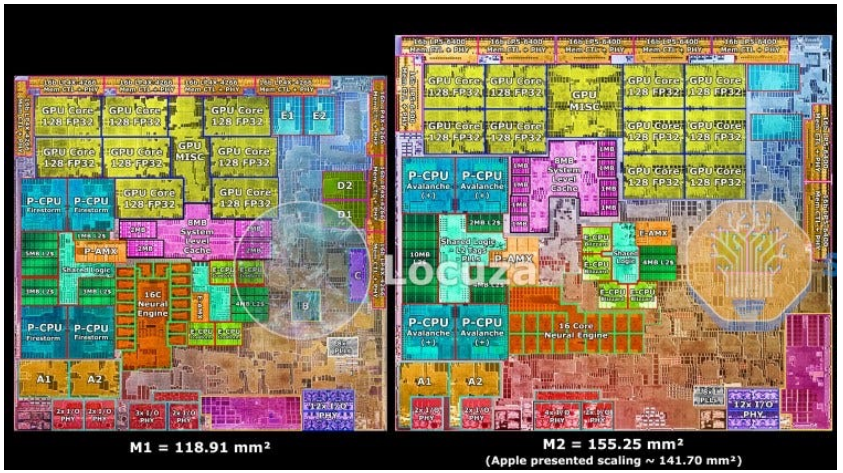
Lecture 2 - Simple Machine Implementations



Apple unveils M2, taking the breakthrough performance and capabilities of M1 even further

Apple announced M2, beginning the next generation of Apple silicon designed specifically for the Mac. Built using second-generation 5-nanometer technology, M2 takes the industry-leading performance per watt of M1 even further with an 18 percent faster CPU, a 35 percent more powerful GPU, and a 40 percent faster Neural Engine.[1]

https://www.apple.com/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities/

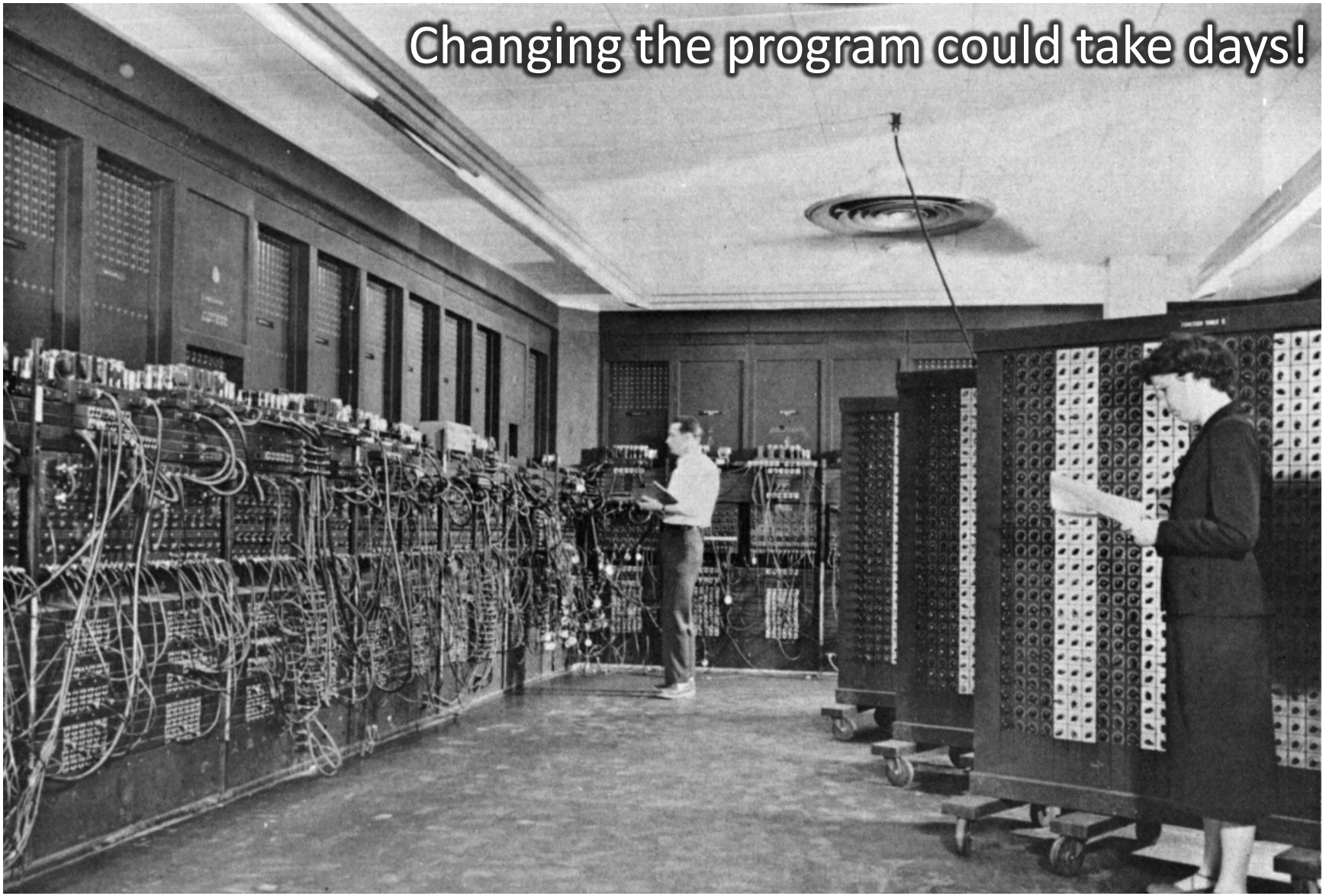


https://www.semianalysis.com/p/apple-m2-die-shot-and-architecture

Last Time in Lecture 1

- Computer Architecture >> ISAs and RTL
  - CS152 is about interaction of hardware and software, and design of appropriate abstraction layers
- Technology and Applications shape Computer Architecture
  - History provides lessons for the future
- First 130 years of CompArch, from Babbage to IBM 360
  - Move from calculators (no conditionals) to fully programmable machines
  - Rapid change started in WWII (mid-1940s), move from electro-mechanical to pure electronic processors
- Cost of software development becomes a large constraint on architecture (need compatibility)

ENIAC



[Public Domain, US Army Photo]

EDVAC

- ENIAC team started discussing stored-program concept to speed up programming and simplify machine design
- John von Nuemann was consulting at UPenn and typed up ideas in “First Draft of a report on EDVAC”
- Herman Goldstine circulated the draft June 1945 to many institutions, igniting interest in the stored-program idea
  - But also, ruined chances of patenting it
  - Report falsely gave sole credit to von Neumann for the ideas
  - Maurice Wilkes was excited by report and decided to come to US workshop on building computers
- Later, in 1948, modifications to ENIAC allowed it to run in stored-program mode, but 6x slower than hardwired
  - Due to I/O limitations, this speed drop was not practically significant and improvement in productivity made it worthwhile
- EDVAC eventually built and (mostly) working in 1951
  - Delayed by patent disputes with university

Cambridge EDSAC (1949)

- Maurice Wilkes came back from workshop in US and set about building a stored-program computer in Cambridge
- EDSAC used mercury-delay line storage to hold up to 1024 words (512 initially) of 17 bits (+1 bit of padding in delay line)
- Two's-complement binary arithmetic
- Accumulator ISA with self-modifying code for indexing
- David Wheeler, who earned the world's first computer science PhD, invented the subroutine ("Wheeler jump") for this machine
  - Users built a large library of useful subroutines
- UK's first commercial computer, LEO-I (Lyons Electronic Office), was based on EDSAC, ran business software in 1951
  - Software for LEO was still running in the 1980s in emulation on ICL mainframes!
- EDSAC-II (1958) was first machine with microprogrammed control unit

IBM 701 (1952)

- IBM's first commercial scientific computer
- Main memory was 72 William's Tubes, each 1Kib, for total of 2048 words of 36 bits each
  - Memory cycle time of 12μs
- Accumulator ISA with multipler/quotient register
- 18-bit/36-bit numbers in sign-magnitude fixed-point
- Misquote from Thomas Watson Sr/Jr:

"I think there is a world market for maybe five computers"

- Actually TWJr said at shareholder meeting:

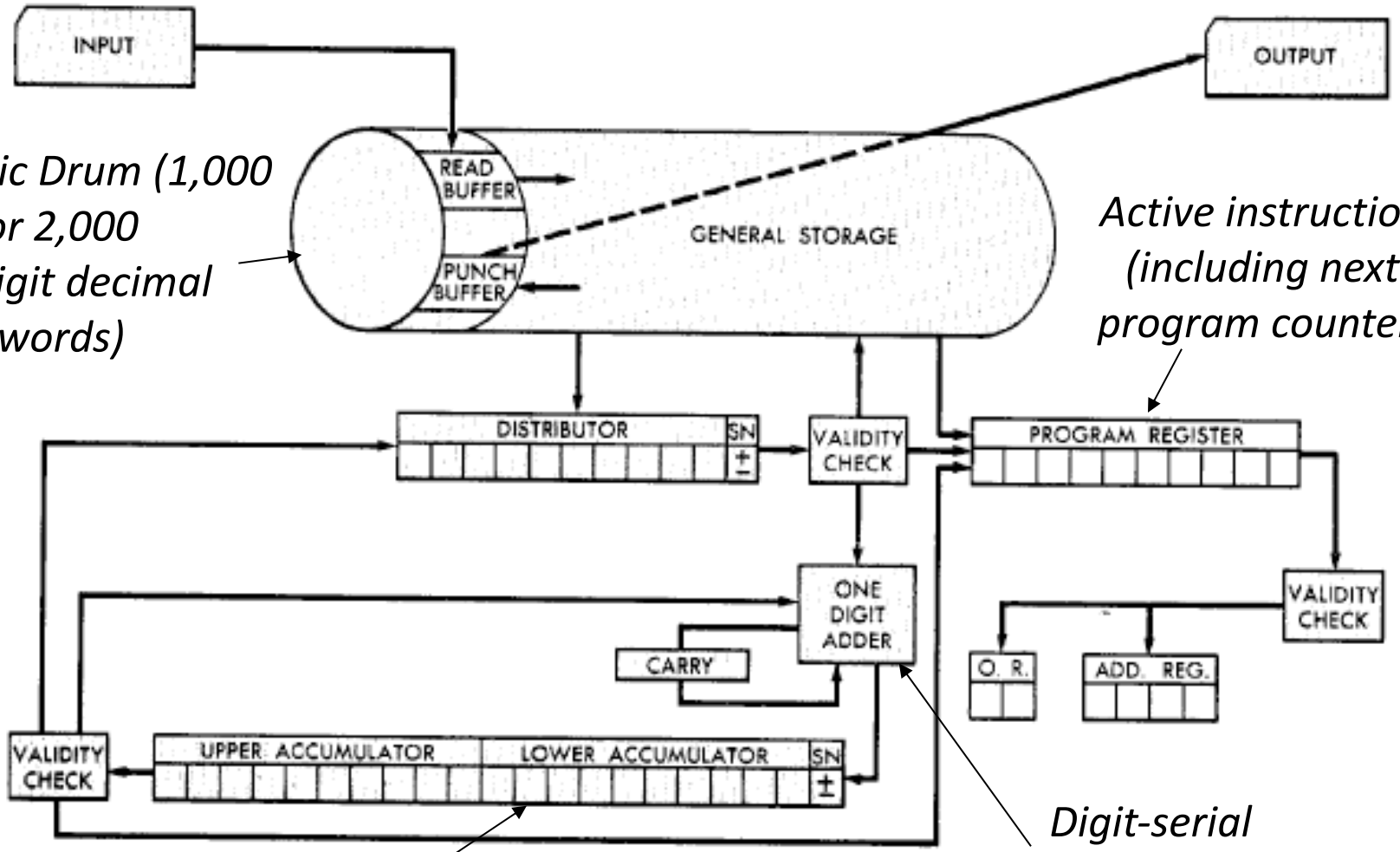
"as a result of our trip [selling the 701]***, on which we expected to get orders for five machines, we came home with orders for 18."***

IBM 650 (1953)

- The first mass-produced computer
- Low-end system with drum-based memory and digit serial ALU
- Almost 2,000 produced

[Cushing Memorial Library and Archives, Texas A&M, Creative Commons Attribution 2.0 Generic]

IBM 650 Architecture



Magnetic Drum (1,000 or 2,000 10-digit decimal words)

Active instruction (including next program counter)

20-digit accumulator

Digit-serial ALU

[From 650 Manual, © IBM]

IBM's Big Bet: 360 Architecture

- By early 1960s, IBM had several incompatible families of computer:

  701 → 7094
  650 → 7074
  702 → 7080
  1401 → 7010

- Each system had its own
  - Instruction set
  - I/O system and secondary storage (magnetic tapes, drums and disks)
  - assemblers, compilers, libraries,...
  - market niche (business, scientific, real time, ...)

IBM 360: A General-Purpose Register (GPR) Machine

- Processor State
  - 16 General-Purpose 32-bit Registers
    - may be used as index and base register
    - Register 0 has some special properties
  - 4 Floating Point 64-bit Registers
  - A Program Status Word (PSW)
    - PC, Condition codes, Control flags
- A 32-bit machine with 24-bit addresses
  - But no instruction contains a 24-bit address!
- Data Formats
  - 8-bit bytes, 16-bit half-words, 32-bit words, 64-bit double-words

The IBM 360 is why bytes are 8-bits long today!

IBM 360: Initial Implementations

	Model 30	. . .	*Model 70*
Storage	8K - 64 KB		256K - 512 KB
Datapath	8-bit		64-bit
Circuit Delay	30 nsec/level		5 nsec/level
Local Store	Main Store		Transistor Registers
Control Store	Read only 1microsec		Dedicated circuits

IBM 360 instruction set architecture (ISA) completely hid the underlying technological differences between various models.

Milestone: The first true ISA designed as portable hardware-software interface!

With minor modifications it still survives today!

IBM Mainframes survive until today

[z15, 2020, 14nm technology, 17 layers of metal, 696 sq mm]

IBM z

IBM

z15 Processor Design Summary

Micro-Architecture

- 12 cores per CP-chip
- 5.2GHz
- More than 9.1 Billion Transistors

- Cache/TLB Improvements:
  - 128KB I$ + 128KB D$
  - L2 I/D$ (4MB)
  - 256MB L3 Cache
  - 12 Concurrent L2$ Misses
  - Enhanced D$ hardware prefetcher
  - 512 entry 2-gig TLB2

- Pipeline Optimizations:
  - SHL/LHS avoidance improvements
  - Issue/Execution side swap on long running VecOps
  - Larger Global Completion Table
  - Larger Issue Queues
  - New Mapper design
  - BFU latency/throughput improvements

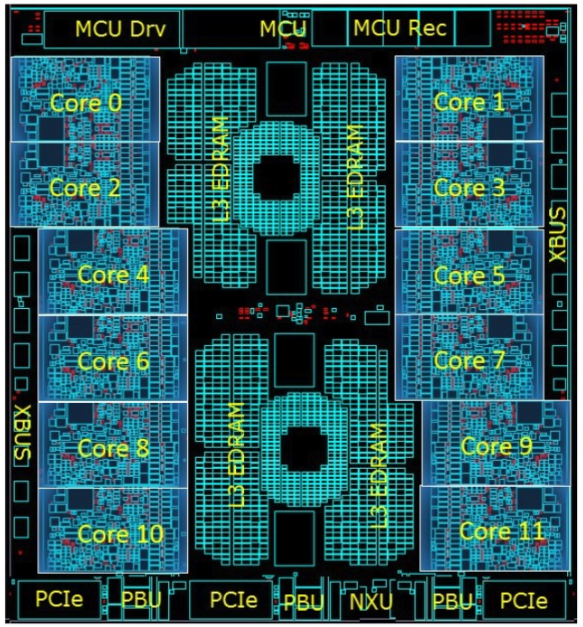
- Branch Prediction Improvements:
  - 16K enhanced BTB1 design
  - New Tage based PHT predictor
  - Improved call/return predictor

Architecture

- Secure Execution
- 38 new instructions for:
  - GPR based logical operations
  - Accelerators
  - Vector search & shifting
  - Vector load/store reversed
  - Vector 2x bandwidth loads
  - Conversions & more!

Accelerators

- On Chip Deflate (gzip)
- On Core Modulo Arithmetic (ECC)
- On Core sort/merge acceleration



Instruction Set Architecture (ISA)

- The contract between software and hardware
- Typically described by giving all the programmer-visible state (registers + memory) plus the semantics of the instructions that operate on that state
- IBM 360 was first line of machines to separate ISA from implementation (aka. *microarchitecture*)
- Many implementations possible for a given ISA
  - E.g., Soviets built code-compatible clones of the IBM360, as did Amdahl after he left IBM.
  - E.g.2., AMD, Intel, VIA processors run the AMD64 ISA
  - E.g.3: many cellphones use the ARM ISA with implementations from many different companies including Apple, Qualcomm, Samsung, Huawei, etc.
- We use RISC-V as standard ISA in class (www.riscv.org)
  - Many companies and open-source projects build RISC-V implementations

ISA to Microarchitecture Mapping

- ISA often designed with particular microarchitectural style in mind, e.g.,

  Accumulator $\Rightarrow$ hardwired, unpipelined

  CISC $\Rightarrow$ microcoded

  RISC $\Rightarrow$ hardwired, pipelined

  VLIW $\Rightarrow$ fixed-latency in-order parallel pipelines

- But can be implemented with any microarchitectural style
  - Intel Ivy Bridge: hardwired pipelined CISC (x86) machine (with some microcode support)
  - Apple M1 (native ARM ISA, emulates x86 in software)
  - Spike: Software-interpreted RISC-V machine
  - **This lecture: a microcoded RISC-V machine**

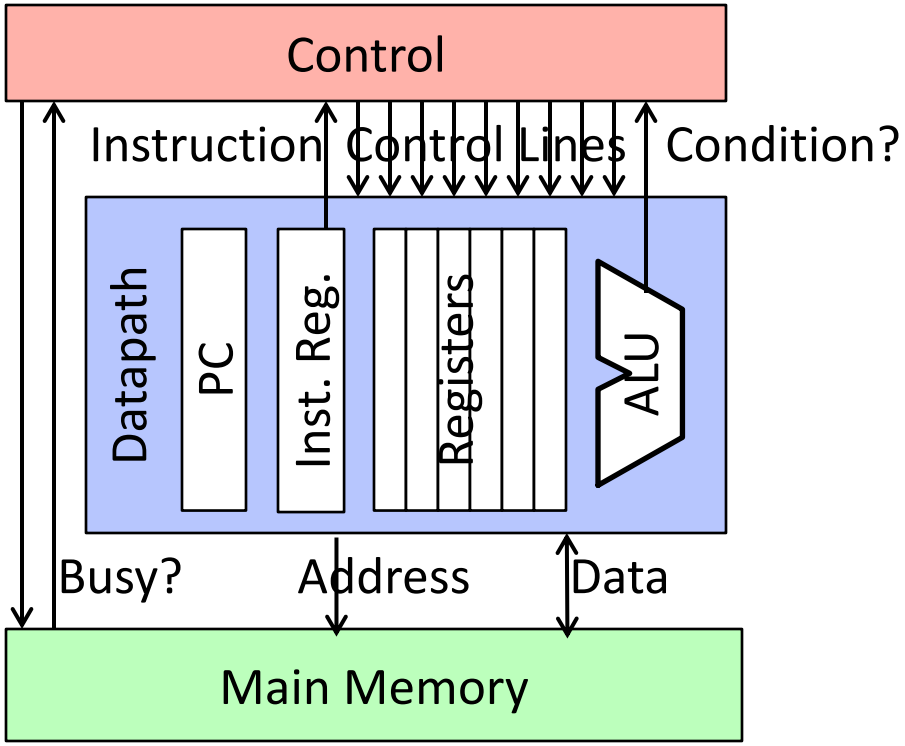
Why Learn Microprogramming?

- To show how to build very small processors with complex ISAs
- To help you understand where CISC* machines came from
- Because still used in common machines (x86, IBM360, PowerPC)
- As a gentle introduction into machine structures
- To help understand how technology drove the move to RISC*

** "CISC"/"RISC" names much newer than style of machines they refer to.*

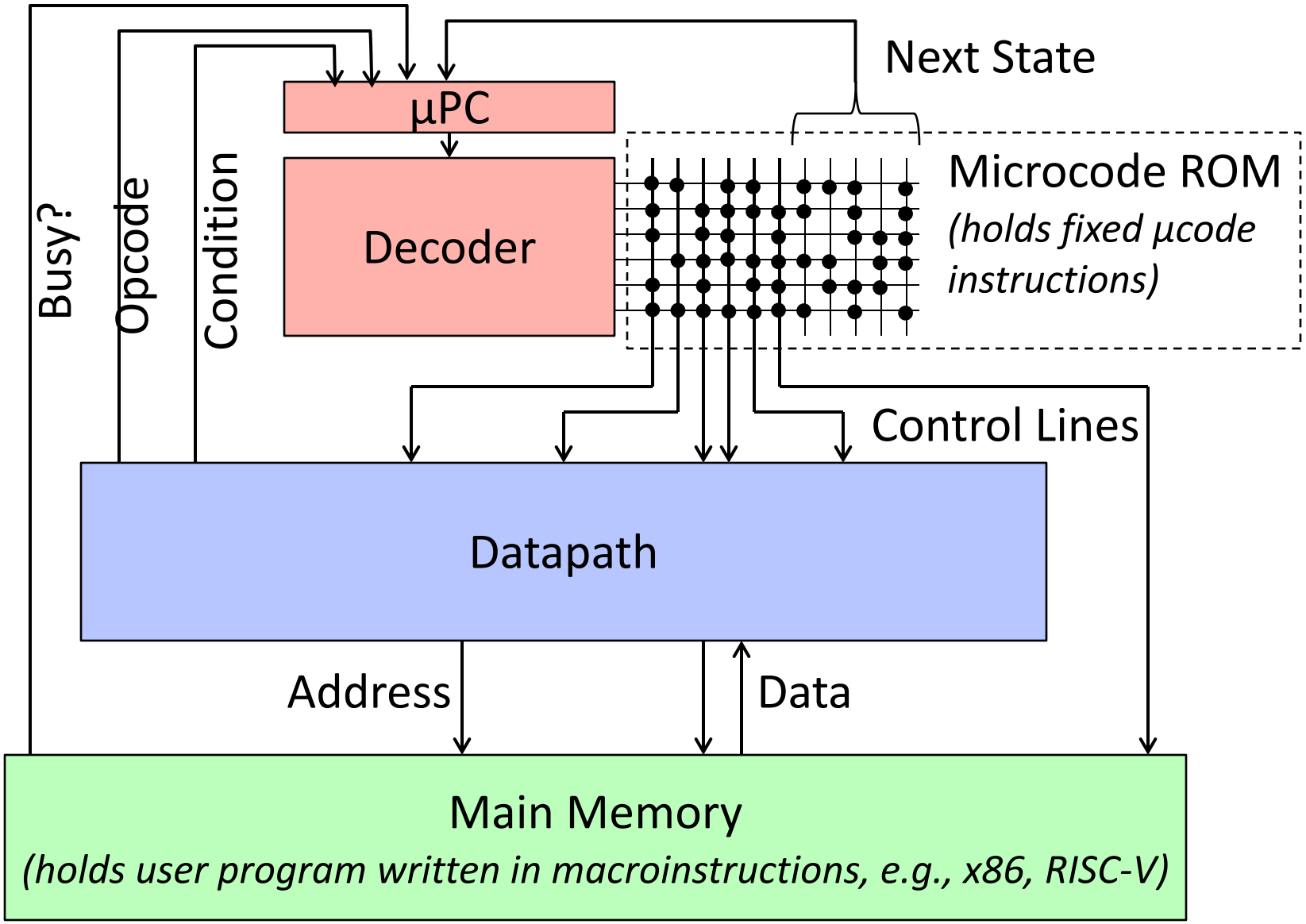
Control versus Datapath

- Processor designs can be split between *datapath*, where numbers are stored and arithmetic operations computed, and *control*, which sequences operations on datapath



- Biggest challenge for early computer designers was getting control circuitry correct
- Maurice Wilkes invented the idea of microprogramming to design the control unit of a processor for EDSAC-II, 1958
  - Foreshadowed by Babbage's "Barrel" and mechanisms in earlier programmable calculators

Microcoded CPU



Technology Influence

- When microcode appeared in 1950s, different technologies for:
  - Logic: Vacuum Tubes
  - Main Memory: Magnetic cores
  - Read-Only Memory: Diode matrix, punched metal cards, …
- Logic very expensive compared to ROM or RAM
- ROM cheaper than RAM
- ROM much faster than RAM

RISC-V ISA

- New fifth-generation RISC design from UC Berkeley
- Realistic & complete ISA, but open & small
- Not over-architected for a certain implementation style
- Both 32-bit (RV32) and 64-bit (RV64) address-space variants
- Designed for multiprocessing
- Efficient instruction encoding
- Easy to subset/extend for education/research
- RISC-V spec available on Foundation website and github
- Increasing momentum with industry adoption

- Please see CS61C for RISC-V ISA review: https://cs61c.org/

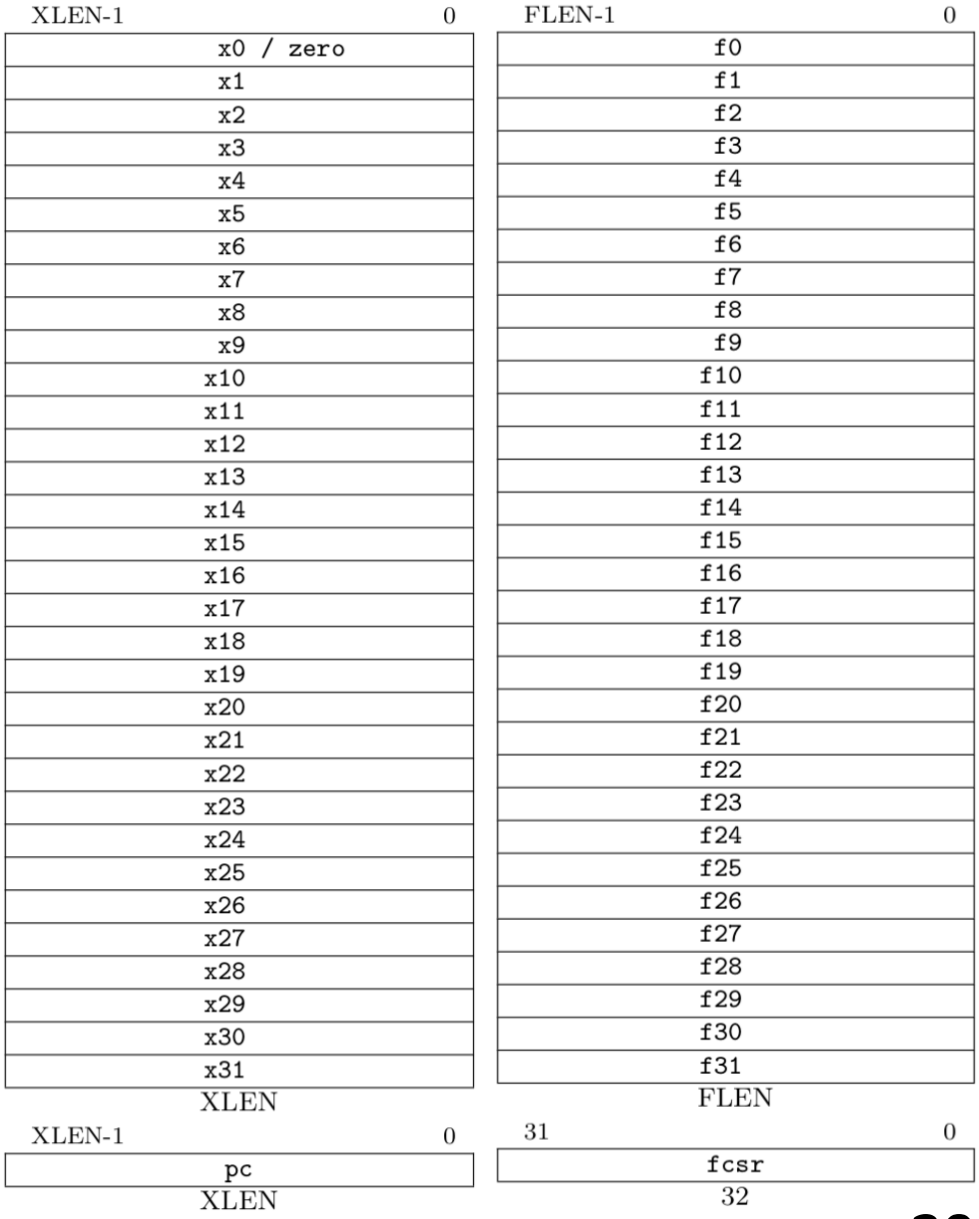
RV32I Processor State

Program counter (**pc**)

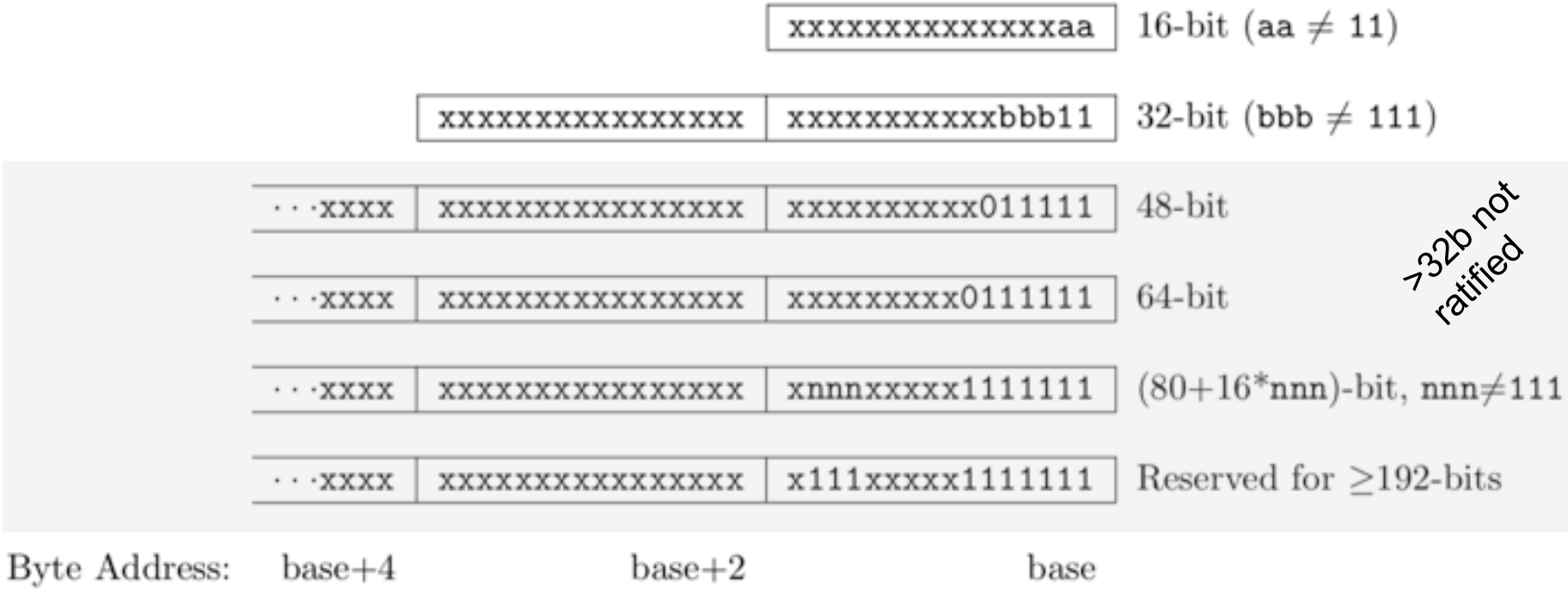
32x32-bit integer registers (**x0-x31**)
• **x0** always contains a 0

32 floating-point (FP) registers (**f0-f31**)
• each can contain a single- or double-precision FP value (32-bit or 64-bit IEEE FP)

FP status register (**fcsr**), used for FP rounding mode & exception reporting

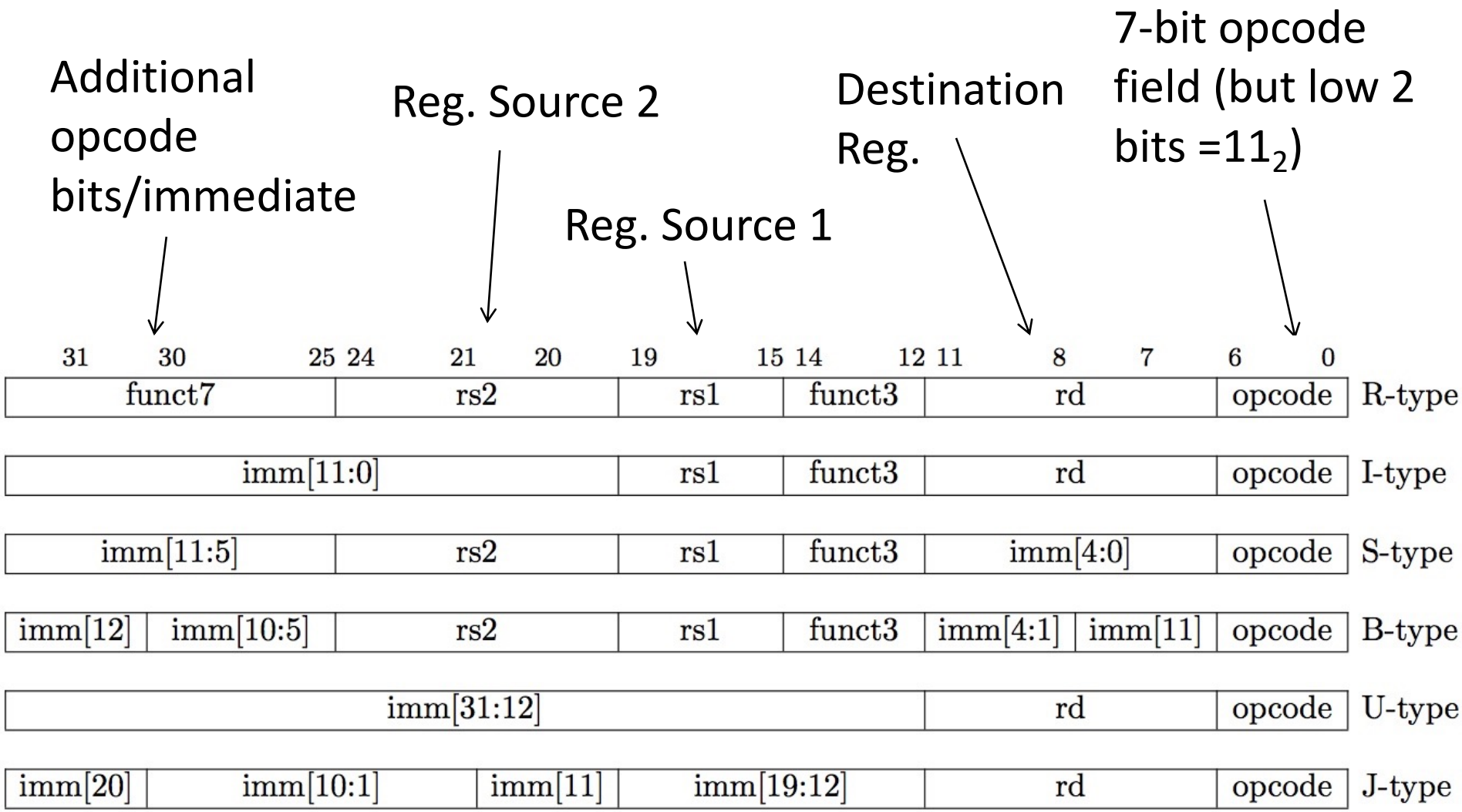


RISC-V Instruction Encoding

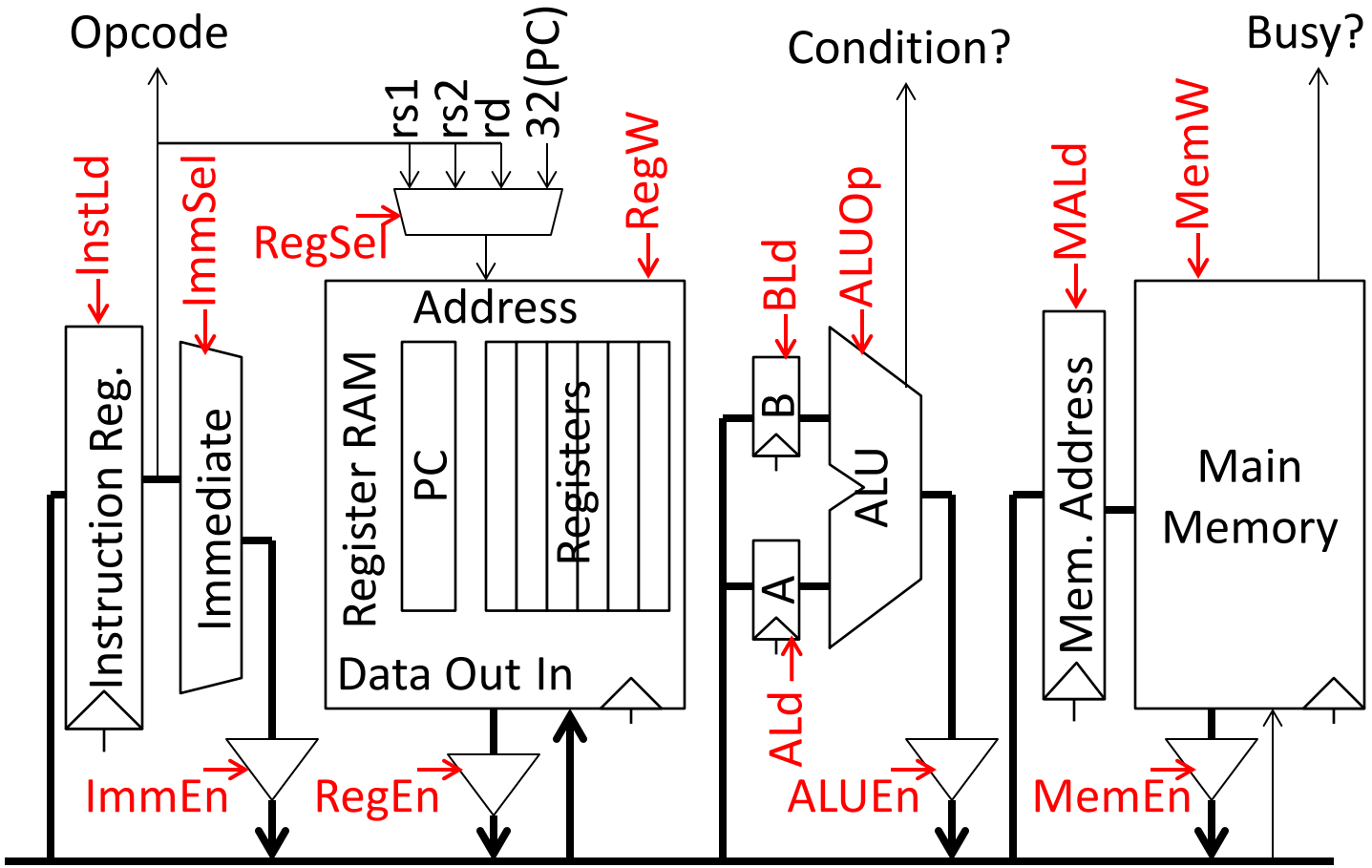


- Can support variable-length instructions.
- Base instruction set (RV32) always has fixed 32-bit instructions lowest two bits = 11_2
- All branches and jumps have targets at 16-bit granularity (even in base ISA where all instructions are fixed 32 bits)

RISC-V Instruction Formats



Single-Bus Datapath for Microcoded RISC-V



Microinstructions written as register transfers:

- MA:=PC means RegSel=PC; RegW=0; RegEn=1; MALd=1
- B:=Reg[rs2] means RegSel=rs2; RegW=0; RegEn=1; BLd=1
- Reg[rd]:=A+B means ALUop=Add; ALUEn=1; RegSel=rd; RegW=1

RISC-V Instruction Execution Phases

- Instruction Fetch
- Instruction Decode
- Register Fetch
- ALU Operations
- *Optional* Memory Operations
- *Optional* Register Writeback
- Calculate Next Instruction Address

Microcode Sketches (1)

Instruction Fetch:

MA,A:=PC

PC:=A+4

wait for memory

IR:=Mem

dispatch on opcode

ALU:

A:=Reg[rs1]

B:=Reg[rs2]

Reg[rd]:=ALUOp(A,B)

goto instruction fetch

ALUI:

A:=Reg[rs1]

B:=ImmI //Sign-extend 12b immediate

Reg[rd]:=ALUOp(A,B)

goto instruction fetch

Microcode Sketches (2)

LW:

A:=Reg[rs1]
B:=ImmI //Sign-extend 12b immediate
MA:=A+B
wait for memory
Reg[rd]:=Mem
goto instruction fetch

JAL:

Reg[rd]:=A // Store return address (A:=PC from fetch)
A:=A-4 // Recover PC (PC incremented in fetch)
B:=ImmJ // Jump-style immediate
PC:=A+B // (Alternative: PC:=A+B-4)
goto instruction fetch

Branch:

A:=Reg[rs1]
B:=Reg[rs2]
if (!ALUOp(A,B)) *goto instruction fetch* //Not taken
A:=PC //Microcode fall through if branch taken
A:=A-4
B:=ImmB// Branch-style immediate
PC:=A+B
goto instruction fetch

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252