Computer Architecture

Spring 2016

Lecture 03: Introduction III

Shuai Wang

Department of Computer Science and Technology

Nanjing University

Today's Lecture

Review of the following topics

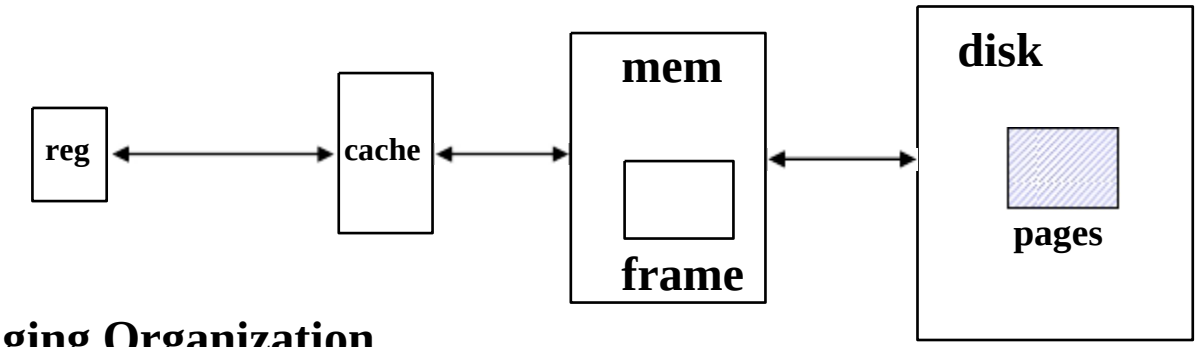
- Pipelining
- Caches
- Virtual memory

Fundamentals of computer design

- Performance
- Cost
- Technology

Basic Issues in VM System Design

- **size of information blocks that are transferred from secondary to main storage (M)**
- **block of information brought into M, and M is full, then some region of M must be released to make room for the new block --> *replacement policy***
- **which region of M is to hold the new block --> *placement policy***
- **missing item fetched from secondary memory only on the occurrence of a fault --> *demand load policy***



Paging Organization

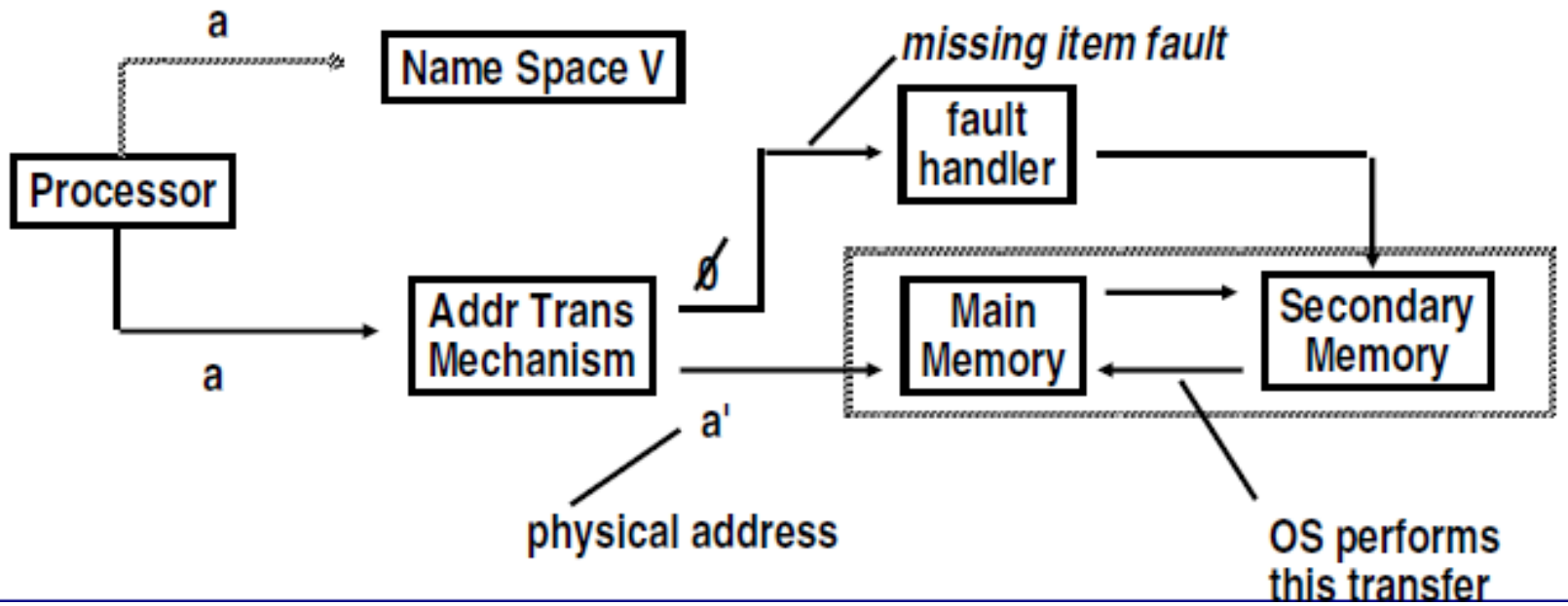
virtual and physical address space partitioned into blocks of equal size

pages

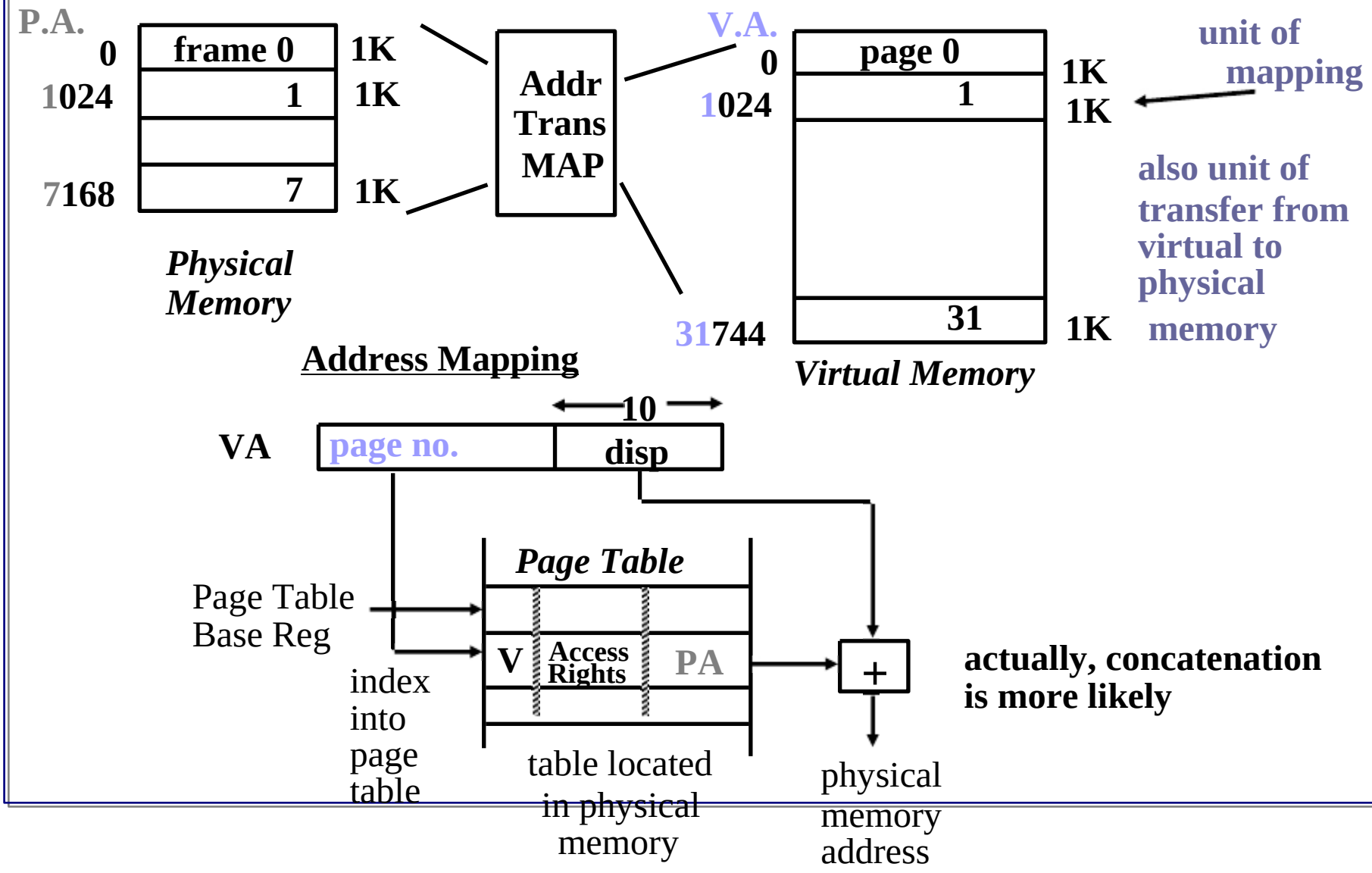
page frames

Address Map

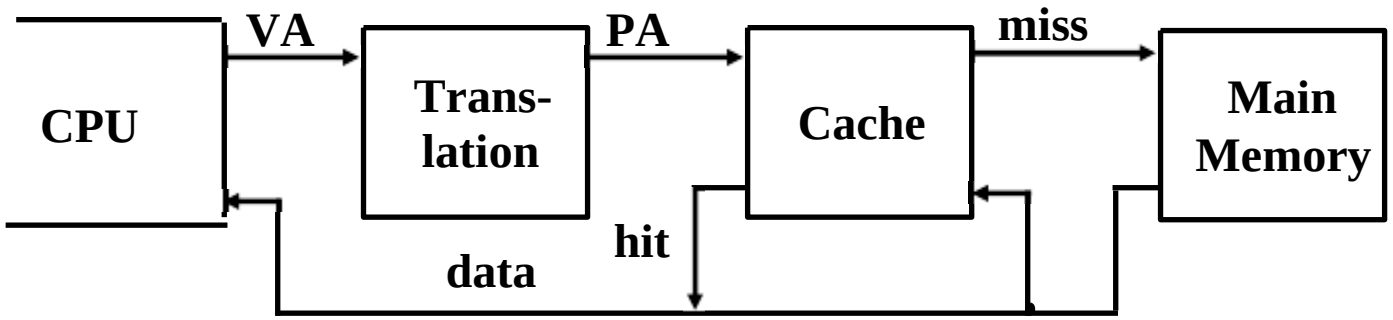
- **V = {0, 1, . . . , n - 1} virtual address space**
- **M = {0, 1, . . . , m - 1} physical address space (m < n)**
- **MAP: V --> M U {0} address mapping function**
  - **MAP(a) = a' if data at virtual address a is present in physical address a' and a' in M**
  - **= 0 if data at virtual address a is not present in M**



Paging Organization



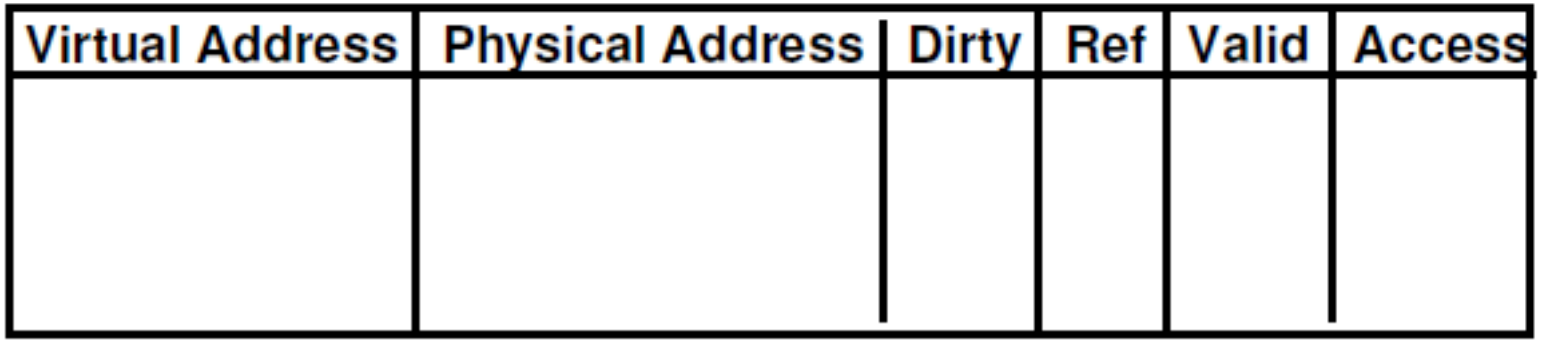
Virtual Address and A Cache



- It takes an extra memory access to translate VA to PA
- This makes cache access very expensive, and this is the "innermost loop" that you want to go as fast as possible
- ASIDE: Why access cache with PA at all? VA caches have a problem!
  - *synonym / alias problem:* two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!
  - for update: must update all cache entries with same physical address or memory becomes inconsistent
  - determining this requires significant hardware: essentially an associative lookup on the physical address tags to see if you have multiple hits or software enforced alias boundary: same lsb of VA & PA > cache size

TLBs

- **A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB***

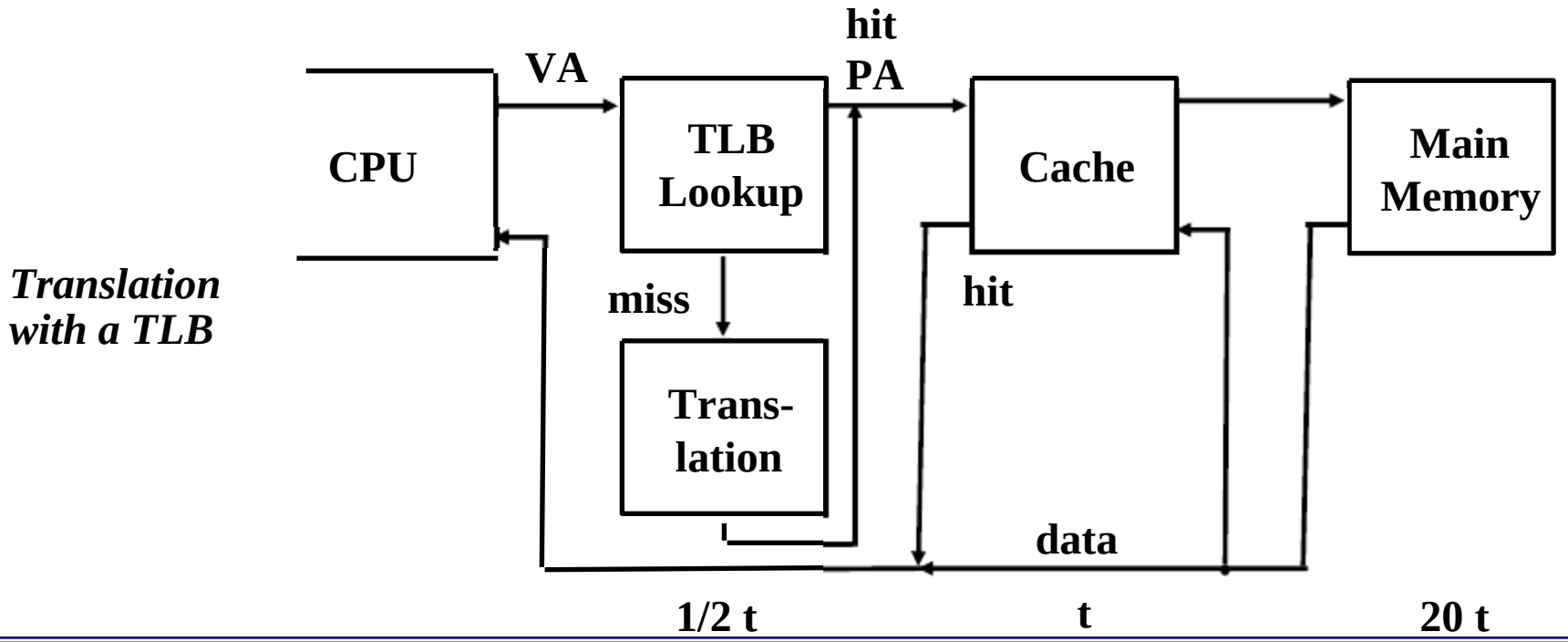


Really just a cache on the page table mappings

TLB access time comparable to cache access time
(much less than main memory access time)

Translation Look-Aside Buffers

- **Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped**
- **TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative**

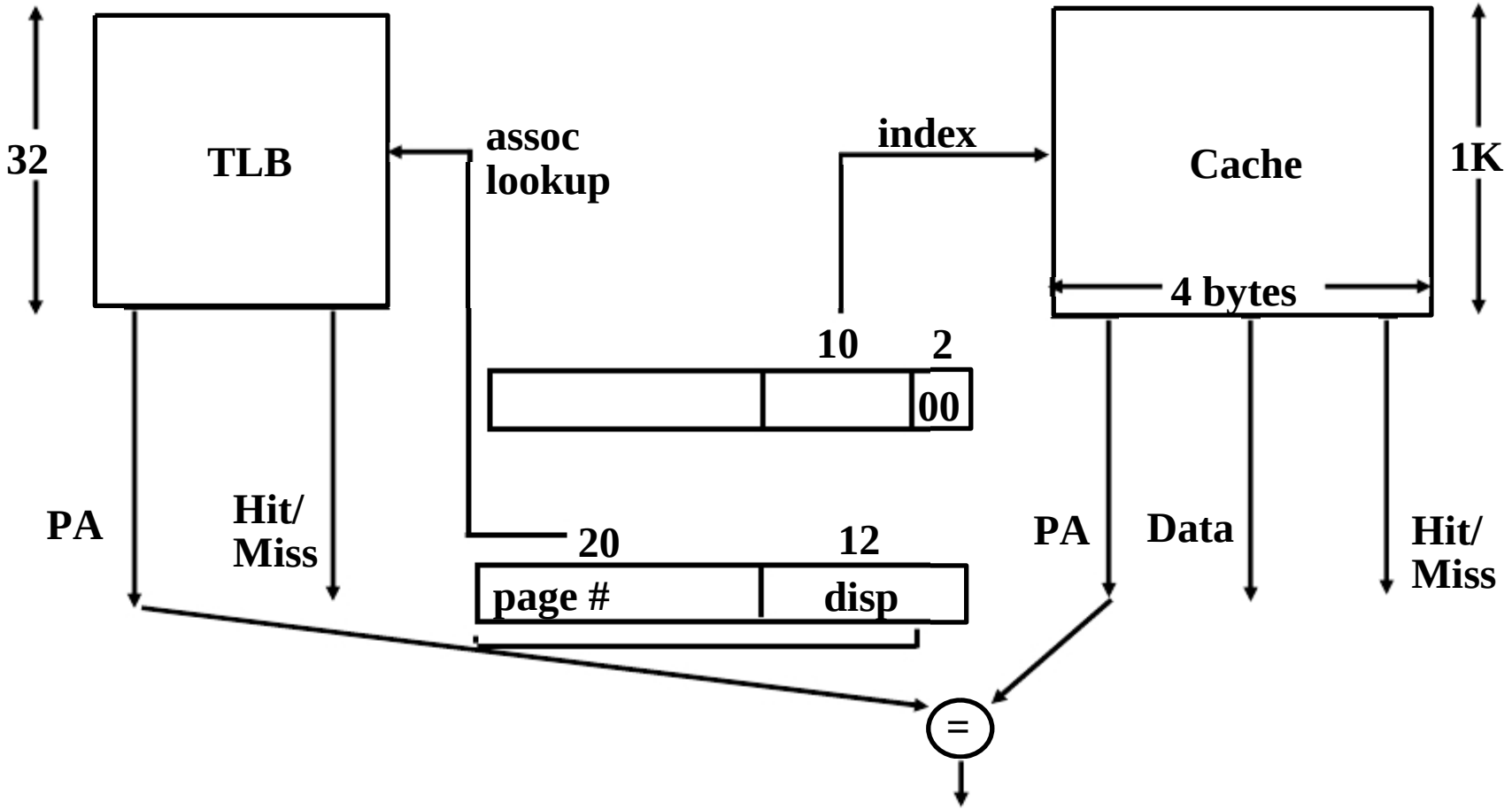


Translation with a TLB

Reducing Translation Time

- Machines with TLBs go one step further to reduce # cycles/cache access
- They overlap the cache access with the TLB access:
  - high order bits of the VA are used to look in the TLB while low order bits are used as index into cache

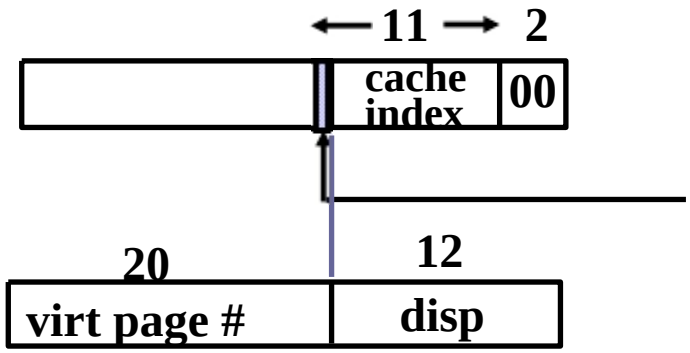
Overlapped Cache & TLB Access



IF cache hit AND (cache tag = PA) then deliver data to CPU
ELSE IF [cache miss OR (cache tag = PA)] and TLB hit THEN
access memory with the PA from the TLB
ELSE do standard VA translation

Problems With Overlapped TLB Access

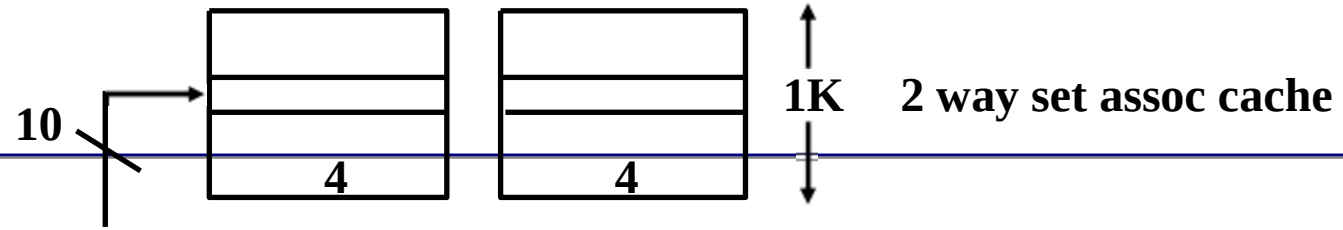
- **Overlapped access only works as long as the address bits used to index into the cache *do not change* as the result of VA translation**
- **This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache**
- **Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:**



This bit is changed by VA translation, but is needed for cache lookup

Solutions:

go to 8K byte page sizes;
go to 2 way set associative cache; or
SW guarantee VA[13]=PA[13]



2 way set assoc cache

Today's Lecture

Review of the following topics

- Pipelining
- Caches
- Virtual memory

Fundamentals of computer design

- Performance
- Cost
- Technology

Performance Measure

- Definition
  - Throughput: total amount of work done in a given time
    - Bigger is better
  - Response time: time between the start and the completion of

$$\textbf{Performance (x)} = \frac{\textbf{1}}{\textbf{Execution\_time (x)}}$$

- "X is n times faster than Y" means

$$n = \frac{\textbf{Performance (X)}}{\textbf{Performance (Y)}} = \frac{\textbf{Execution\_time (Y)}}{\textbf{Execution\_time (X)}}$$

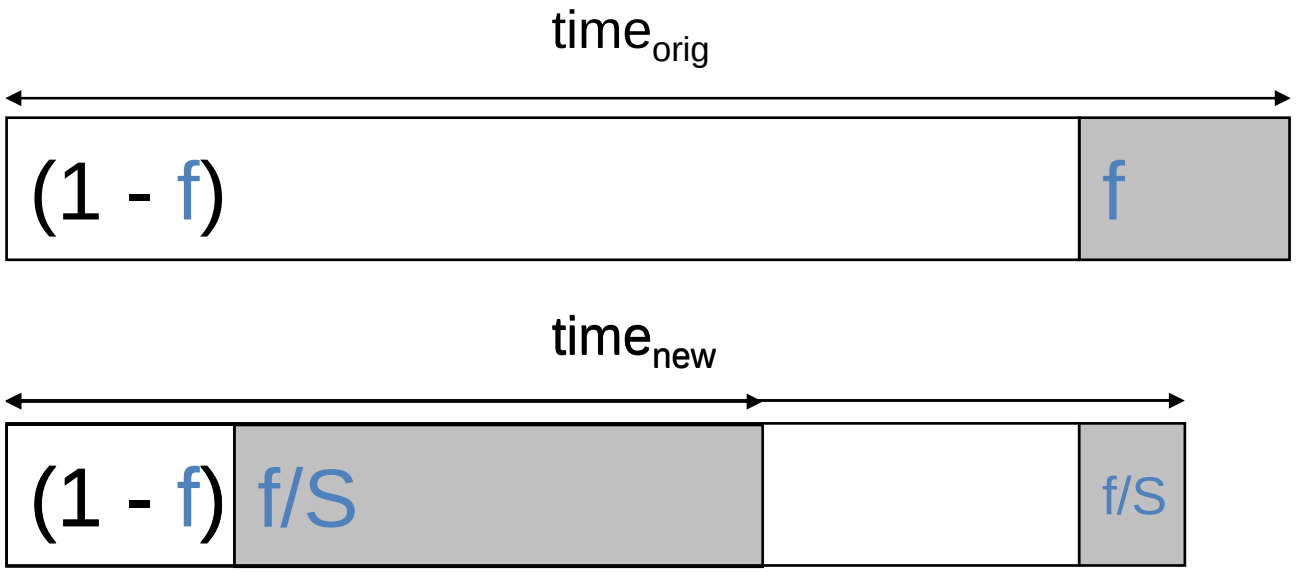
Amdahl's Law

Speedup = $\text{time}_{\text{without enhancement}}$ / $\text{time}_{\text{with enhancement}}$

An enhancement speeds up fraction f of a task by factor S

$$\text{time}_{\text{new}} = \text{time}_{\text{orig}} \cdot ((1-f) + f/S)$$

$$S_{\text{overall}} = 1 / ((1-f) + f/S)$$



Aspects of CPU Performance (CPU Law)

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set	X	X	
Organization		X	X
Technology			X

Cycles Per Instruction (Throughput)

"Average Cycles per Instruction"

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

"Instruction Frequency"

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$

Example: Calculating CPI

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix of instruction types in program

Example: Branch Stall Impact

- Assume CPI = 1.0 ignoring branches
- Assume solution was stalling for 3 cycles
- If 30% branch, Stall 3 cycles

Op	Freq	Cycles	CPI(i)	(% Time)
Other	**70%**	**1**	**.7**	**(37%)**
Branch	**30%**	**4**	**1.2**	**(63%)**

- => new CPI = 1.9, or almost 2 times slower

Example 2: Speed Up Equation for Pipelining

$$CPI_{pipelined} = \text{Ideal } CPI + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \frac{\text{Ideal } CPI \times \text{Pipeline depth}}{\text{Ideal } CPI + \text{Pipeline stall } CPI} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

For simple RISC pipeline, CPI = 1:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall } CPI} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

Example 3: Evaluating Branch Alternatives (for 1 program)

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Scheduling scheme	*Branch penalty*	*CPI*	*speedup v. stall*
Stall pipeline	3	1.42	1.0
Predict taken	1	1.14	1.26
Predict not taken	1	1.09	1.29
Delayed branch	0.5	1.07	1.31

Conditional & Unconditional = 14%, 65% change PC

Example 4: Dual-port vs. Single-port

- Machine A: Dual ported memory (“Harvard Architecture”)
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

$$SpeedUp_A = Pipeline\ Depth/(1 + 0) \times (clock_{unpipe}/clock_{pipe})$$
$$= Pipeline\ Depth$$

$$SpeedUp_B = Pipeline\ Depth/(1 + 0.4 \times 1) \times (clock_{unpipe}/(clock_{unpipe}/1.05))$$
$$= (Pipeline\ Depth/1.4) \times 1.05$$
$$= 0.75 \times Pipeline\ Depth$$

$$SpeedUp_A / SpeedUp_B = Pipeline\ Depth/(0.75 \times Pipeline\ Depth) = 1.33$$

- Machine A is 1.33 times faster

Averaging Performance Numbers

- *Arithmetic*: times
  - proportional to time
  - e.g., latency

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

- *Harmonic*: rates
  - inversely proportional to time
  - e.g., throughput

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{Rate_i}}$$

- *Geometric*: ratios
  - unit-less quantities
  - e.g., speedups

$$\sqrt[n]{\prod_{i=1}^{n} Ratio_i}$$

Computing Devices Now



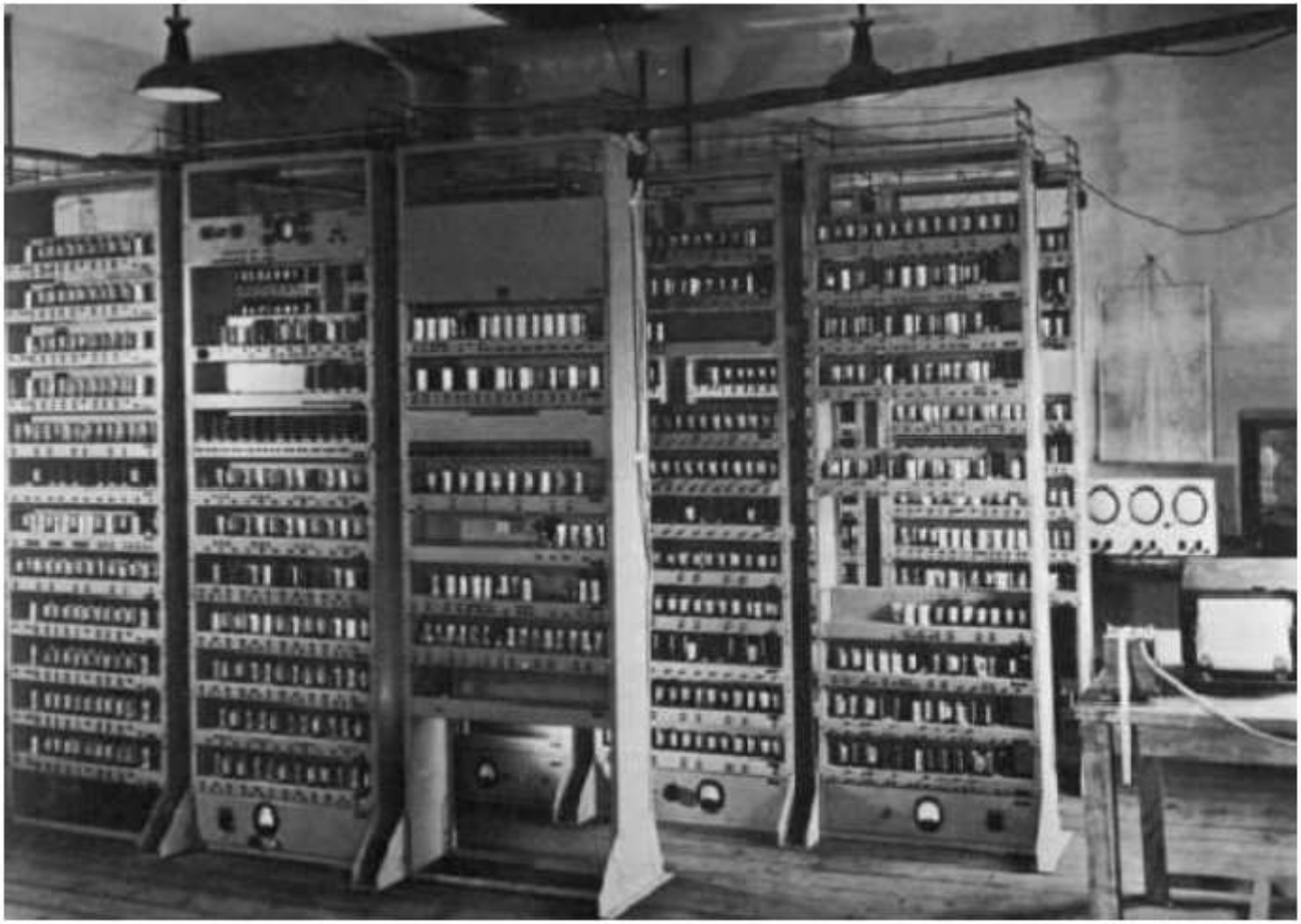


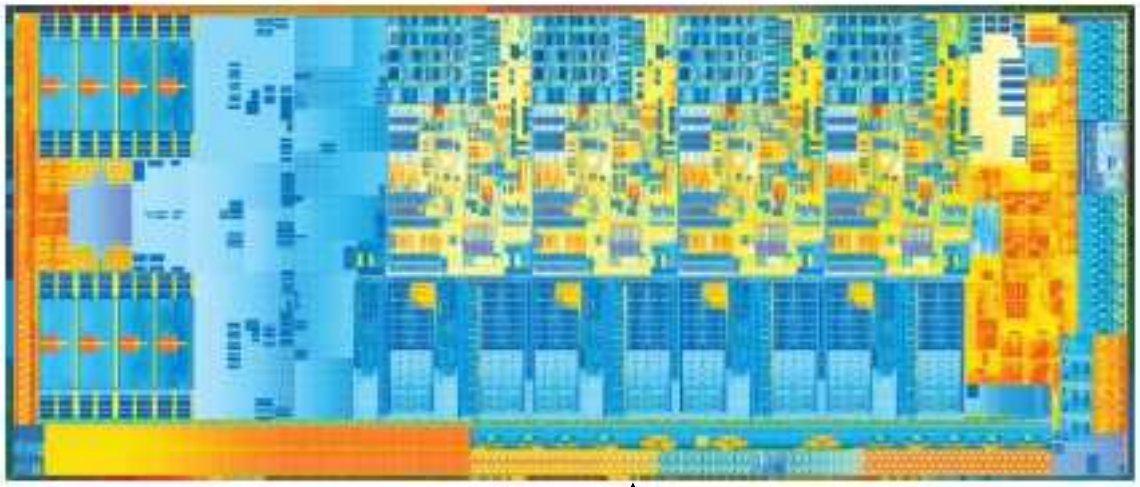







Computing Devices Then…





The Processor you built in the CPU Lab

What you will understand after taking this course

Early Programmable Calculators

- Analog computing was popular in first half of 20th century as digital computing was too expensive
- But during late 30s and 40s, several programmable digital calculators were built (date when operational)
  - Atanasoff Linear Equation Solver (1939)
  - Zuse Z3 (1941)
  - Harvard Mark I (1944)
  - ENIAC (1946)

Atanasoff-Berry Linear Equation Solver (1939)

- Fixed-function calculator for solving up to 29 simultaneous linear equations
- Digital binary arithmetic (50-bit fixed-point words)
- Dynamic memory (rotating drum of capacitors)
- Vacuum tube logic for processing

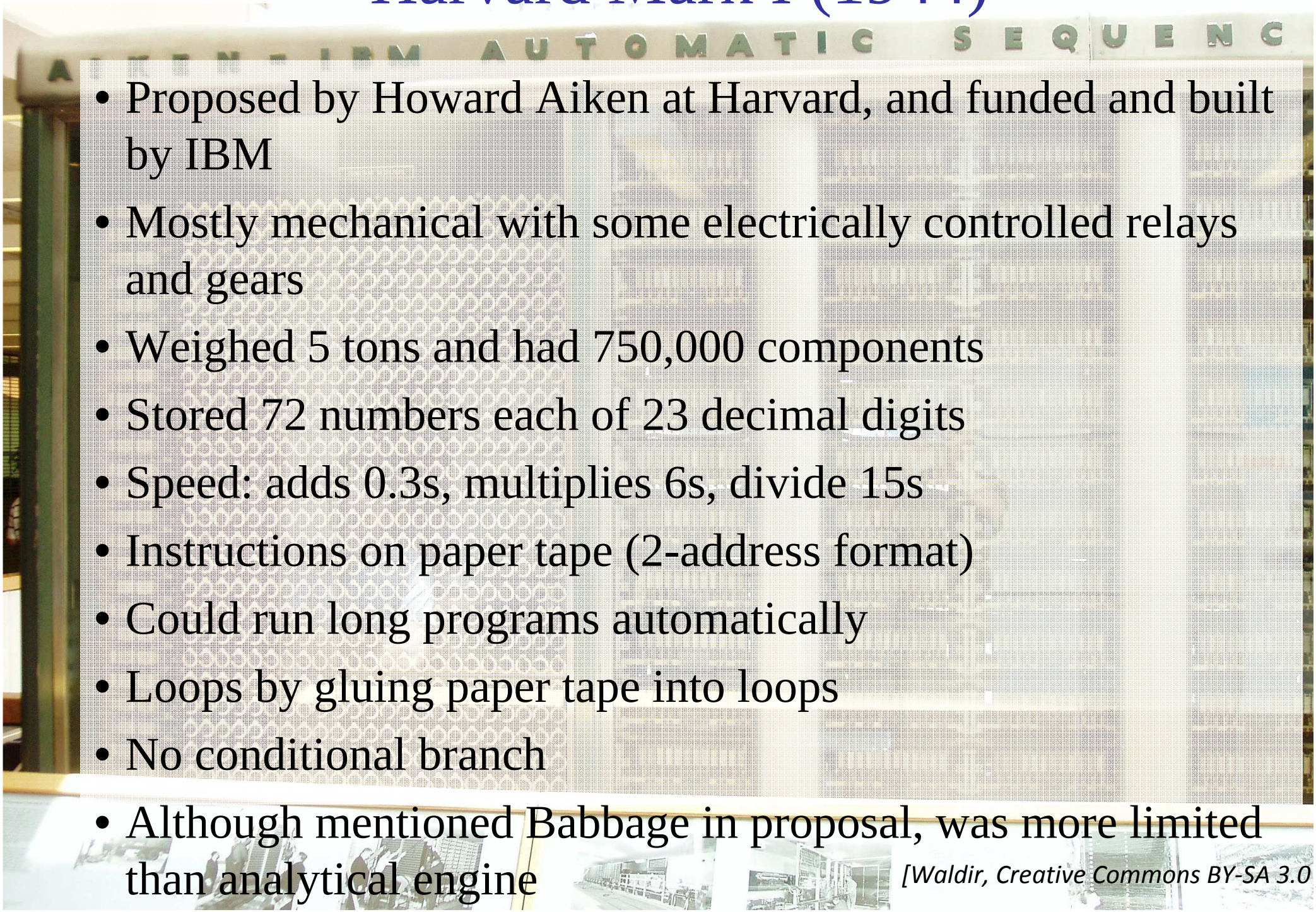
In 1973, Atanasoff was credited as inventor of "automatic electronic digital computer" after patent dispute with Eckert and Mauchly (ENIAC)

Zuse Z3 (1941)

- Built by Konrad Zuse in wartime Germany using 2000 relays
- Had normalized floating-point arithmetic with hardware handling of exceptional values (+/- infinity, undefined)
  - 1-bit sign, 7-bit exponent, 14-bit significand
- 64 words of memory
- Two-stage pipeline 1) fetch&execute 2) writeback
- No conditional branch
- Programmed via paper tape

Replica of the Zuse Z3 in the Deutsches Museum, Munich

Harvard Mark I (1944)



- Proposed by Howard Aiken at Harvard, and funded and built by IBM
- Mostly mechanical with some electrically controlled relays and gears
- Weighed 5 tons and had 750,000 components
- Stored 72 numbers each of 23 decimal digits
- Speed: adds 0.3s, multiplies 6s, divide 15s
- Instructions on paper tape (2-address format)
- Could run long programs automatically
- Loops by gluing paper tape into loops
- No conditional branch
- Although mentioned Babbage in proposal, was more limited than analytical engine

Harvard Mark I

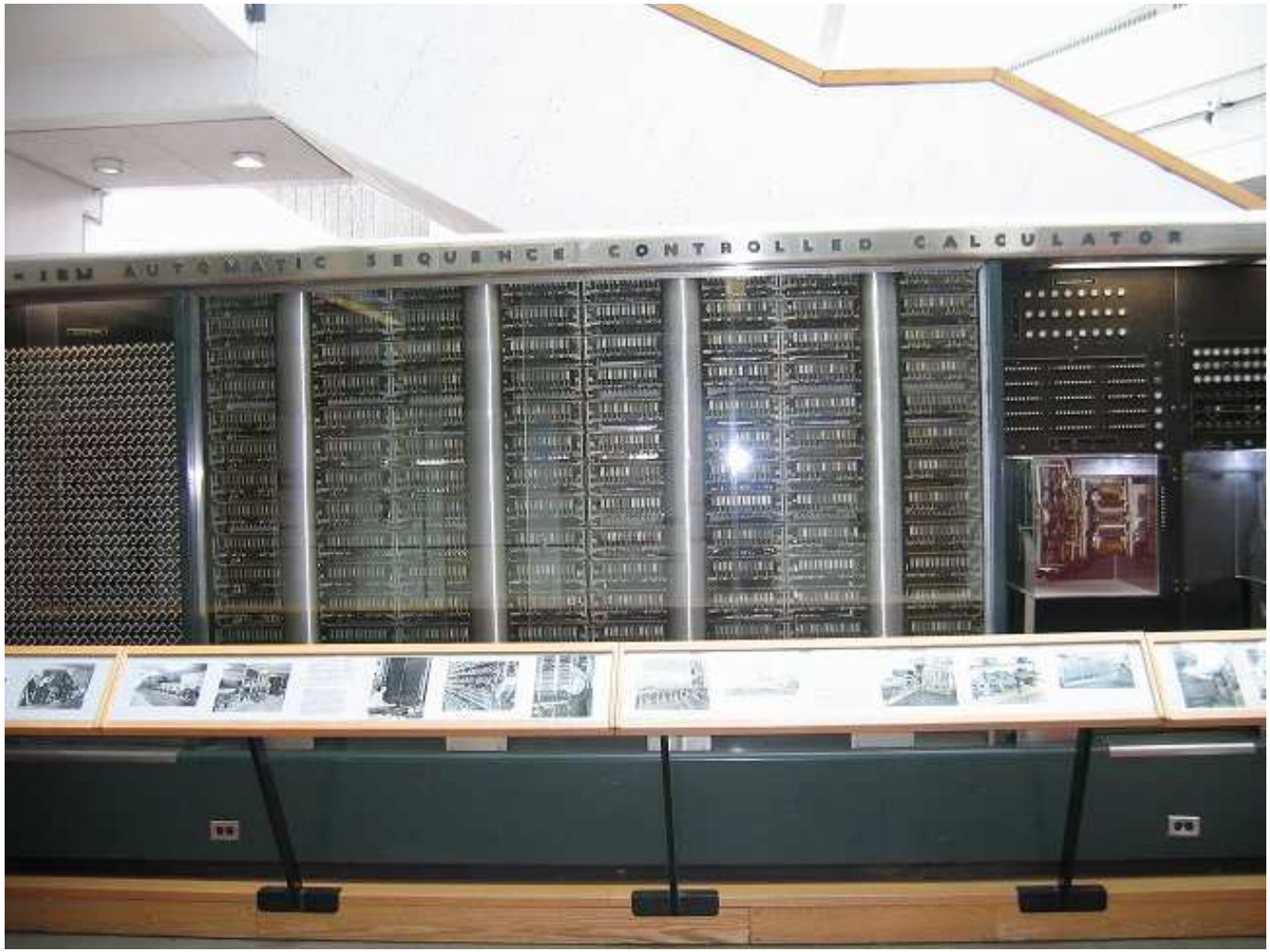


Photo taken in Harvard Science Center

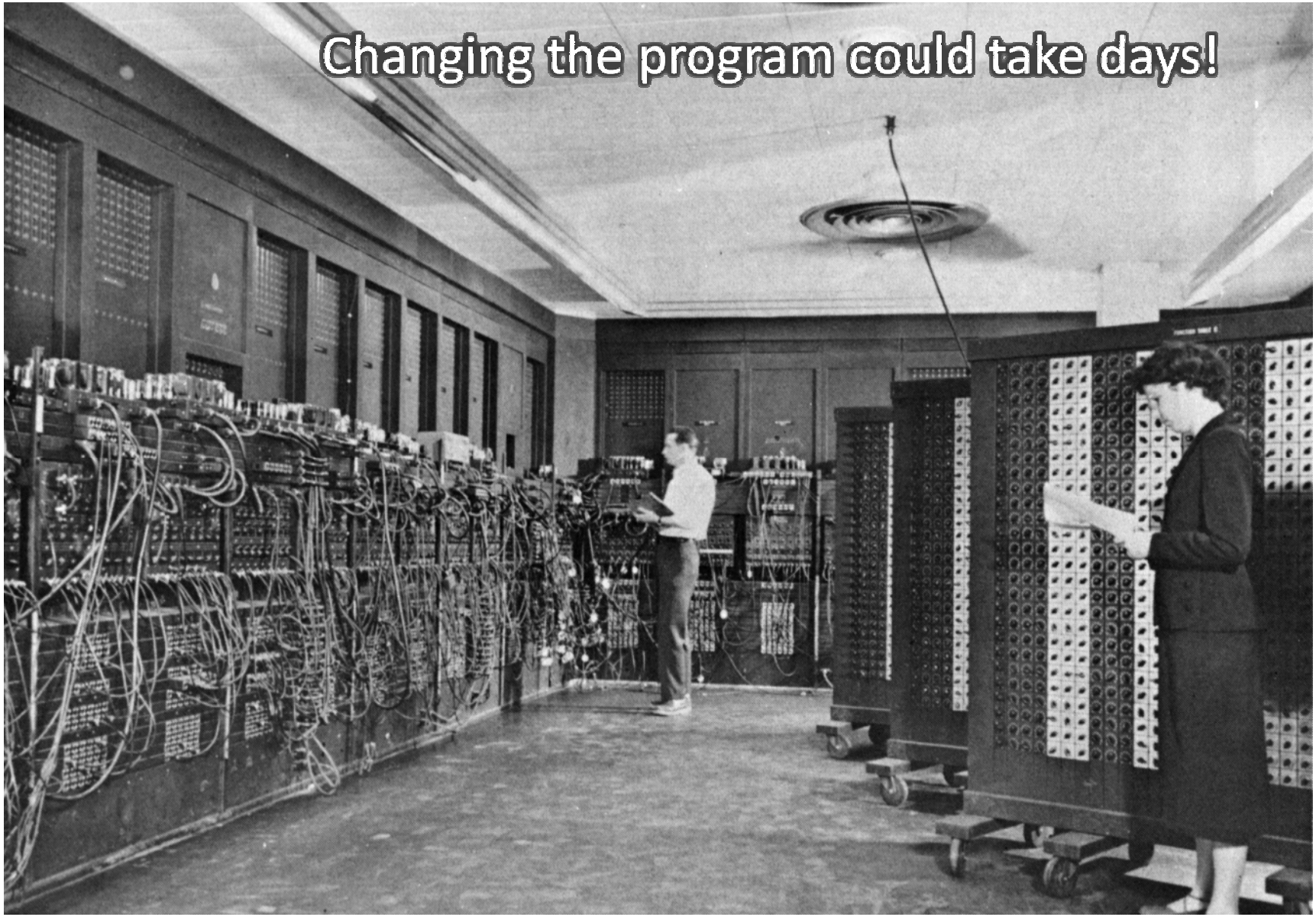
Harvard Mark I

Photo taken in Harvard Science Center

ENIAC (1946)

- First electronic general-purpose computer
- Construction started in secret at UPenn Moore School of Electrical Engineering during WWII to calculate firing tables for US Army, designed by Eckert and Mauchly
- 17,468 vacuum tubes
- Weighed 30 tons, occupied 1800 sq ft, power 150kW
- Twelve 10-decimal-digit accumulators
- Had a conditional branch!
- Programmed by plugboard and switches, time consuming!
- Purely electronic instruction fetch and execution, so fast
  - 10-digit x 10-digit multiply in 2.8ms (2000x faster than Mark-1)
- As a result of speed, it was almost entirely I/O bound
- As a result of large number of tubes, it was often broken (5 days was longest time between failures)

ENIAC



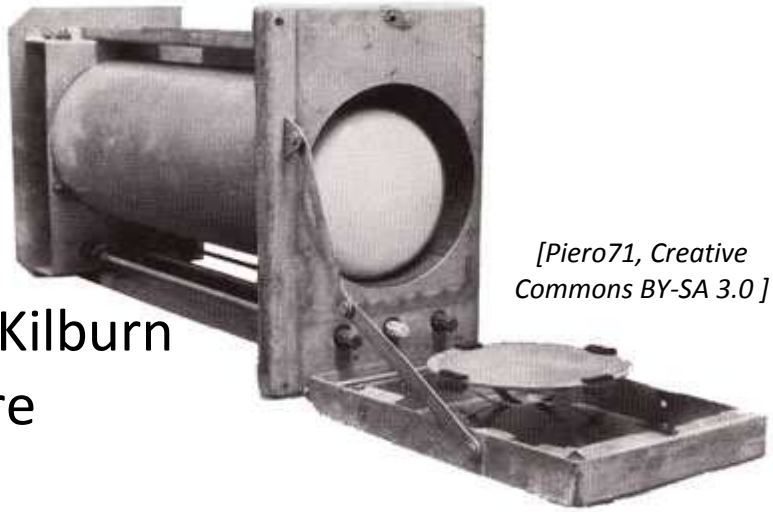
[Public Domain, US Army Photo]

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

Manchester SSEM "Baby" (1948)

- Manchester University group build small-scale experimental machine to demonstrate idea of using cathode-ray tubes (CRTs) for computer memory instead of mercury delay lines
- *Williams-Kilburn Tubes were first random access electronic storage devices*
- 32 words of 32-bits, accumulator, and program counter
- Machine ran world's first stored-program in June 1948
- Led to later Manchester Mark-1 full-scale machine
  - Mark-1 introduced *index* registers
  - Mark-1 commercialized by Ferranti



Williams-Kilburn Tube Store

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

Commercial computers: BINAC (1949) and UNIVAC (1951)

- Eckert and Mauchly left U.Penn after patent rights disputes and formed the Eckert-Mauchly Computer Corporation
- World's first commercial computer was BINAC with two CPUs that checked each other
  - BINAC apparently never worked after shipment to first (only) customer
- Second commercial computer was UNIVAC
  - Used mercury delay-line memory, 1000 words of 12 alpha characters
  - Famously used to predict presidential election in 1952
  - Eventually 46 units sold at >$1M each
  - Often, mistakingly called the IBM UNIVAC

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:

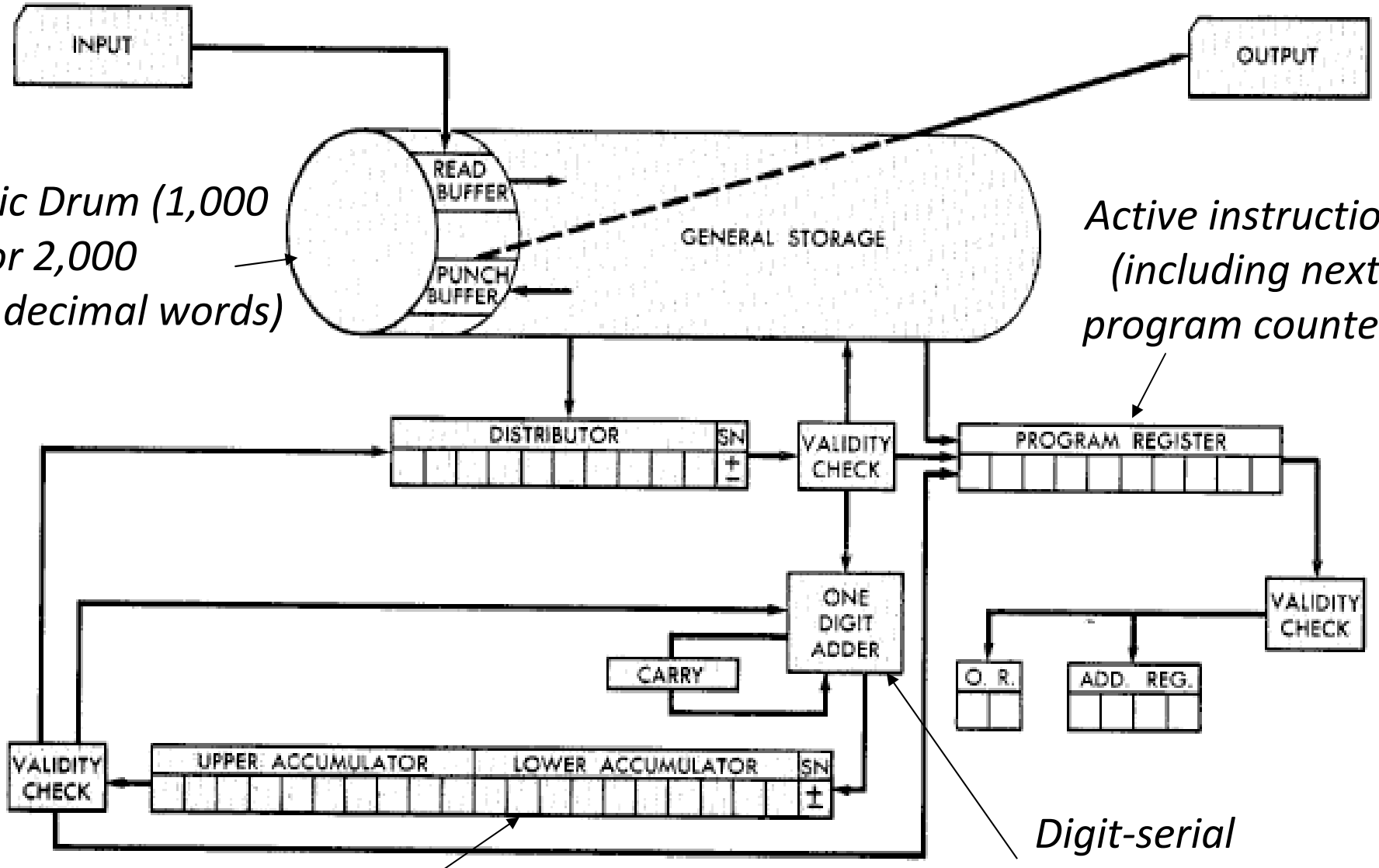
"as a result of our trip [selling the 701]***, on which we expected to get orders for five machines, we came home with orders for 18."***

IBM 650 (1953)

- The first mass-produced computer
- Low-end system with drum-based storage 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)

Digit-serial ALU

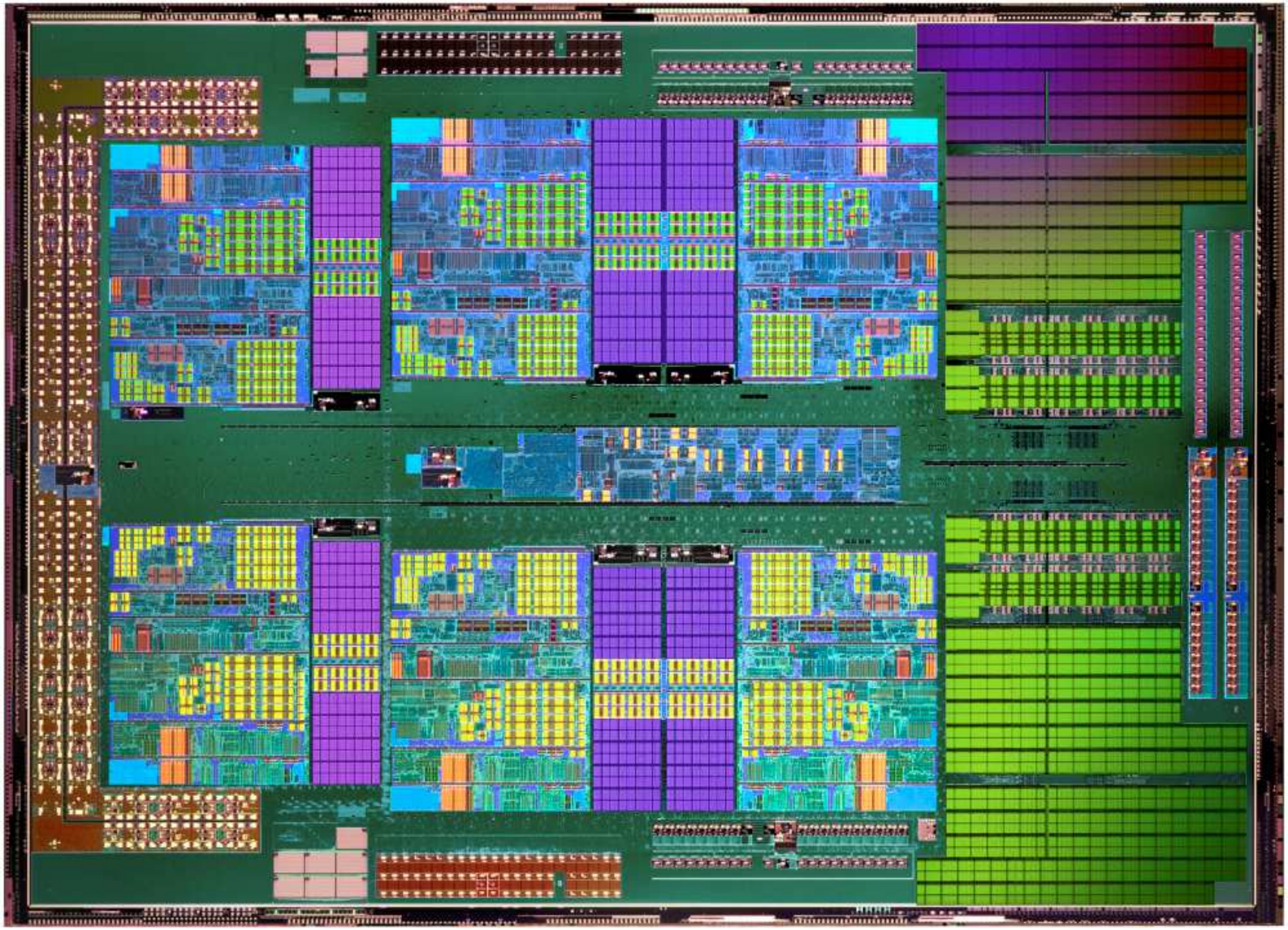
20-digit accumulator

[From 650 Manual, © IBM]

IBM 650 Instruction Set

- Address and data in 10-digit decimal words
- Instructions encode:
  - Two-digit opcode encoded 44 instructions in base instruction set, expandable to 97 instructions with options
  - Four-digit data address
  - Four-digit next instruction address
    - Programmer's arrange code to minimize drum latency!
- Special instructions added to compare value to all words on track

AMD Phenom II X6 Die Shot



John von Neumann

- December 28, 1903 – February 8, 1957
- Hungarian-born American
- Contributions in :
  - Mathematics
  - Computer science
  - Physics
  - Quantum mechanics
  - Game theory
  - ……..

John von Neumann

Gravestone in Princeton Cemetery, NJ, USA

KURT F. GODEL

Gravestone in Princeton Cemetery, NJ, USA

Alan Mathison Turing

- June 23, 1912 – June 7, 1954
- Contributions:
  - Father of Computer Science
  - Mathematician
  - Logician
  - Wartime Codebreaker
  - …..

Alan Mathison Turing

Photo taken in Sackville Park, Manchester, UK

Alan Mathison Turing

Photo taken in Sackville Park, Manchester, UK