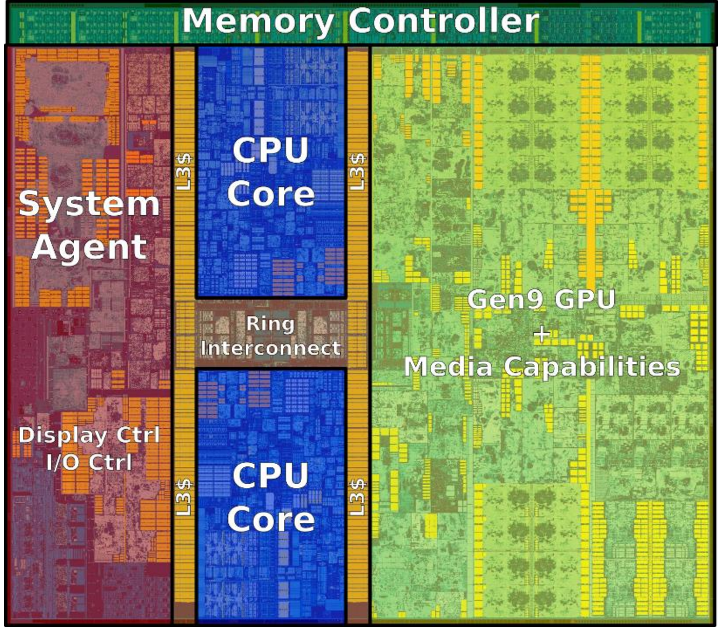
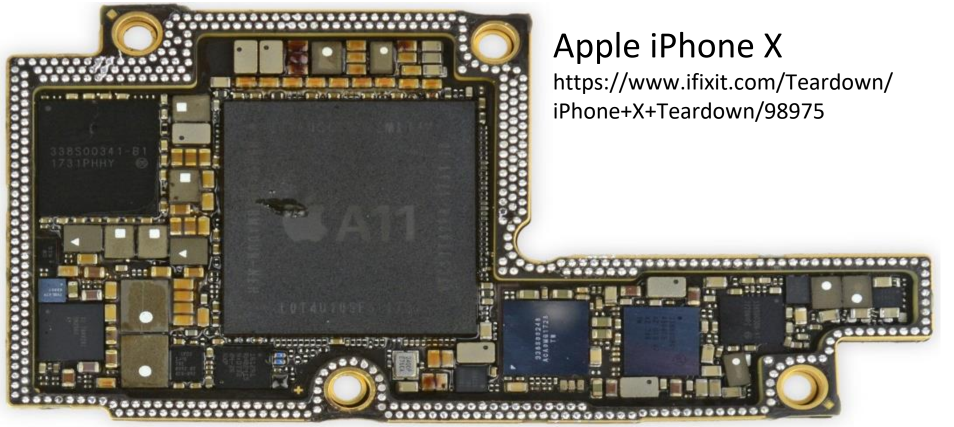
What is this course about?

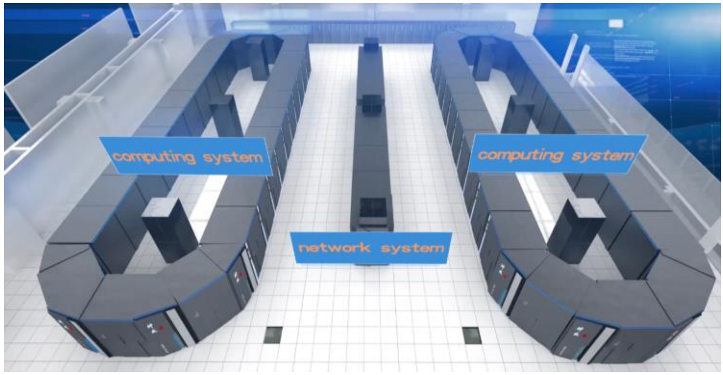
- How the latest microprocessors *work*
- Why they are built that way – and what are the alternatives?
- How you can make software that uses the hardware in the best possible way
- How you can make a compiler that does it for you
- How you can design a computer for *your* problem
- What does a *big* computer look like?
- What are the fundamental big ideas and challenges in computer architecture?
- What is the scope for theory?



Intel Skylake (this laptop) https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)

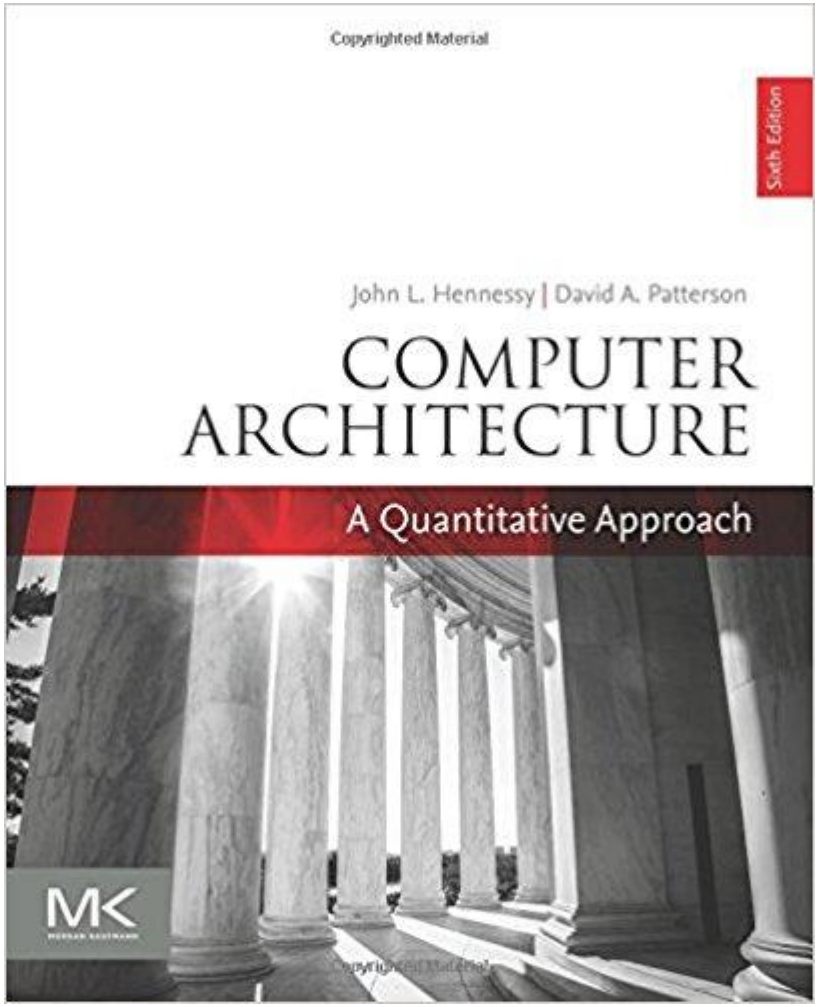


Apple iPhone X https://www.ifixit.com/Teardown/iPhone+X+Teardown/98975



Sunway TaihuLight http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf

This is a textbook-based course



Computer Architecture: A Quantitative Approach (6th Edition)

John L. Hennessy, David A. Patterson

- 936 pages. Morgan Kaufmann (2017)
- ISBN: 9780128119051
- Price: around £62 (shop around!)
- Publisher's companion web site:
  - https://www.elsevier.com/books-and-journals/book-companion/9780128119051
  - Textbook includes some vital introductory material as appendices:
  - Appendix C: tutorial on pipelining (read it NOW)
  - Appendix B: tutorial on memory hierarchy (read it NOW)
- Further appendices (some in book, some online) cover more advanced material (some very relevant to parts of the course), eg
  - Networks
  - Parallel applications
  - Embedded systems
  - Storage systems
  - VLIW
  - Computer arithmetic (esp floating point)
  - Historical perspectives

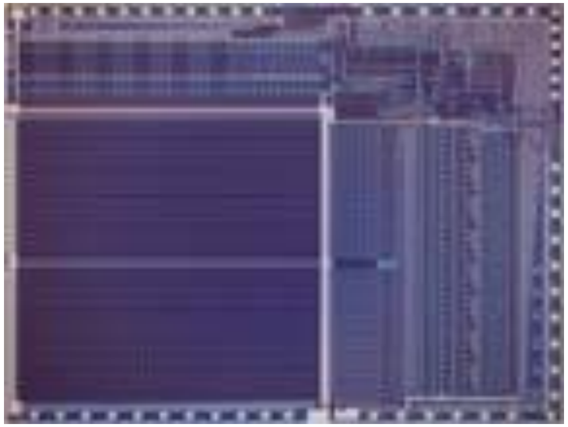
Who are these guys anyway and why should I read their book?

- John Hennessy:
  - Founder, MIPS Computer Systems
  - President (2000-2016), Stanford University
  - Board member, Cisco, chair of Alphabet Inc (parent company of Google)
  - The “godfather of Silicon Valley” (Wikipedia)

RAID-I (1989) consisted of a Sun 4/280 workstation with 128 MB of DRAM, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software.

- David Patterson
  - Leader, Berkeley RISC project
  - RAID (redundant arrays of inexpensive disks)
  - Professor, University of California, Berkeley
  - President of ACM 2004-6
  - Served on Information Technology Advisory Committee to the US

By Peg Skorpinski - Subject of pictures emailed it upon request, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3207893



http://www.cs.berkeley.edu/~pattrsn/Arch/prototypes2.html

RISC-I (1982) Contains 44,420 transistors, fabbed in 5 micron NMOS, with a die area of 77 mm^2, ran at 1 MHz. This chip is probably the first VLSI RISC.

Joint winners of the 2017 ACM Turing Award

“For pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry”

Course organisation

- Lecturer:
  - Paul Kelly – Leader, Software Performance Optimisation research group
    - With help from Jessica Vandebon and Philippos Papaphilippou
- Two two-hour slots per week
- We will use the time flexibly with some sessions in the labs
- In the last couple of lectures we will spend some time on exam preparation

- Assessment:
  - Exam
    - The exam will take place in last week of term
    - The goal of the course is to teach you how to think about computer architecture
    - The exam usually includes some architectural ideas not presented in the lectures
  - Coursework
    - You will be assigned two substantial, laboratory-based exercises
    - You will learn about performance tuning for computationally-intensive kernels
    - You will learn about using simulators, and experimentally evaluating hypotheses to understand system performance
    - You are welcome to bring laptops to class to get started and get help during tutorials (we will go to the DoC labs when necessary)
- **Please do not use computers for anything else during classes**

External Students – Registration for DoC Courses

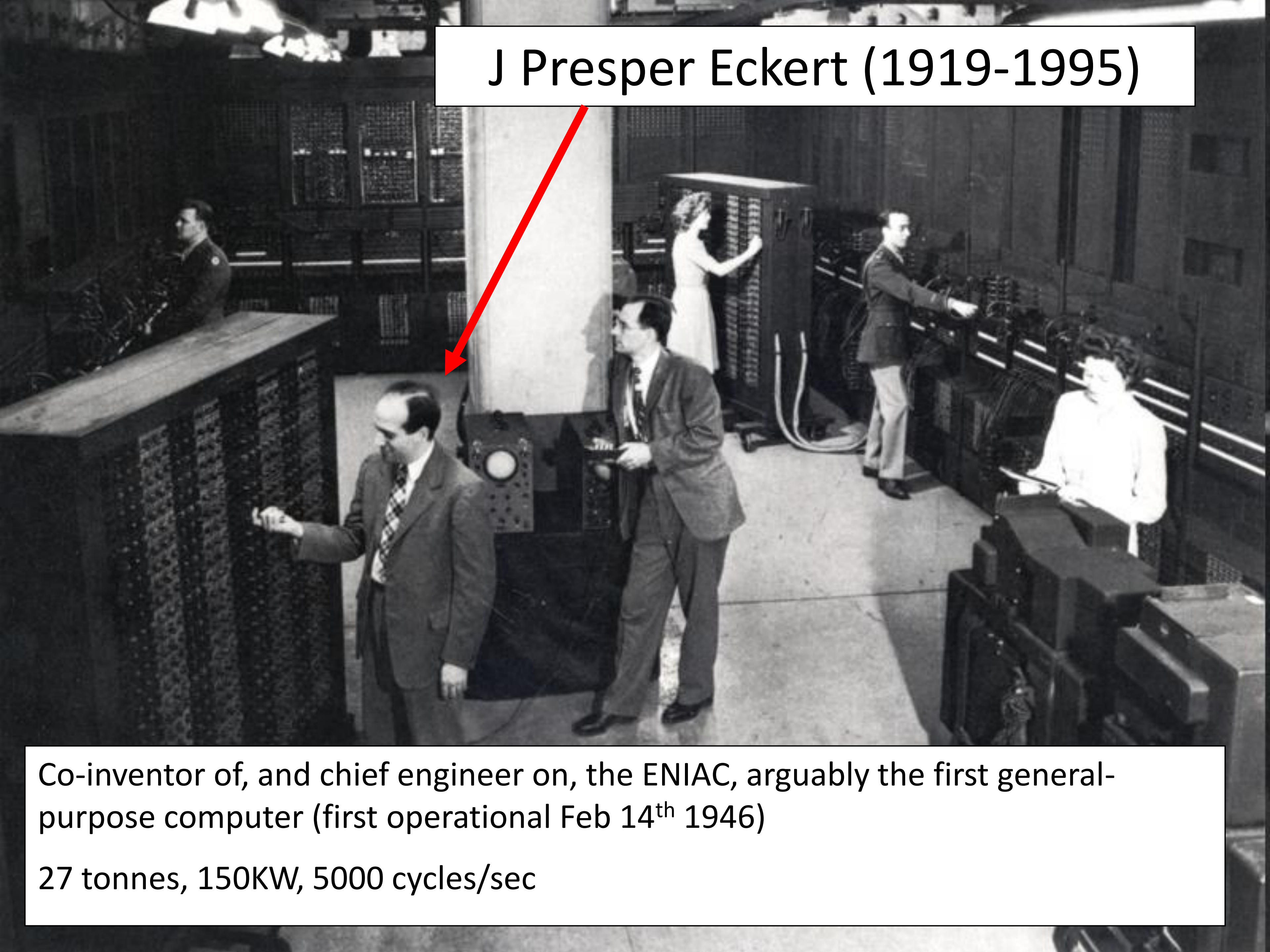
1. Apply at: `https://dbc.doc.ic.ac.uk/externalreg/`
2. Then,
   - Your department's endorser will approve/reject your application
3. If approved,
   - DoC's External Student Liaison will approve/reject your application
4. If approved (again!),
   - Students will get access to DoC resources (DoC account, CATE, . . .)
   - No access after a few days? Check status of approval and contact relevant person(s)

Key Dates

- Exams for DoC $3^{rd}/4^{th}$ yr. Spring courses take place at the end of the Term in week 11
- Registration for credit opens end January and closes mid February for Spring courses

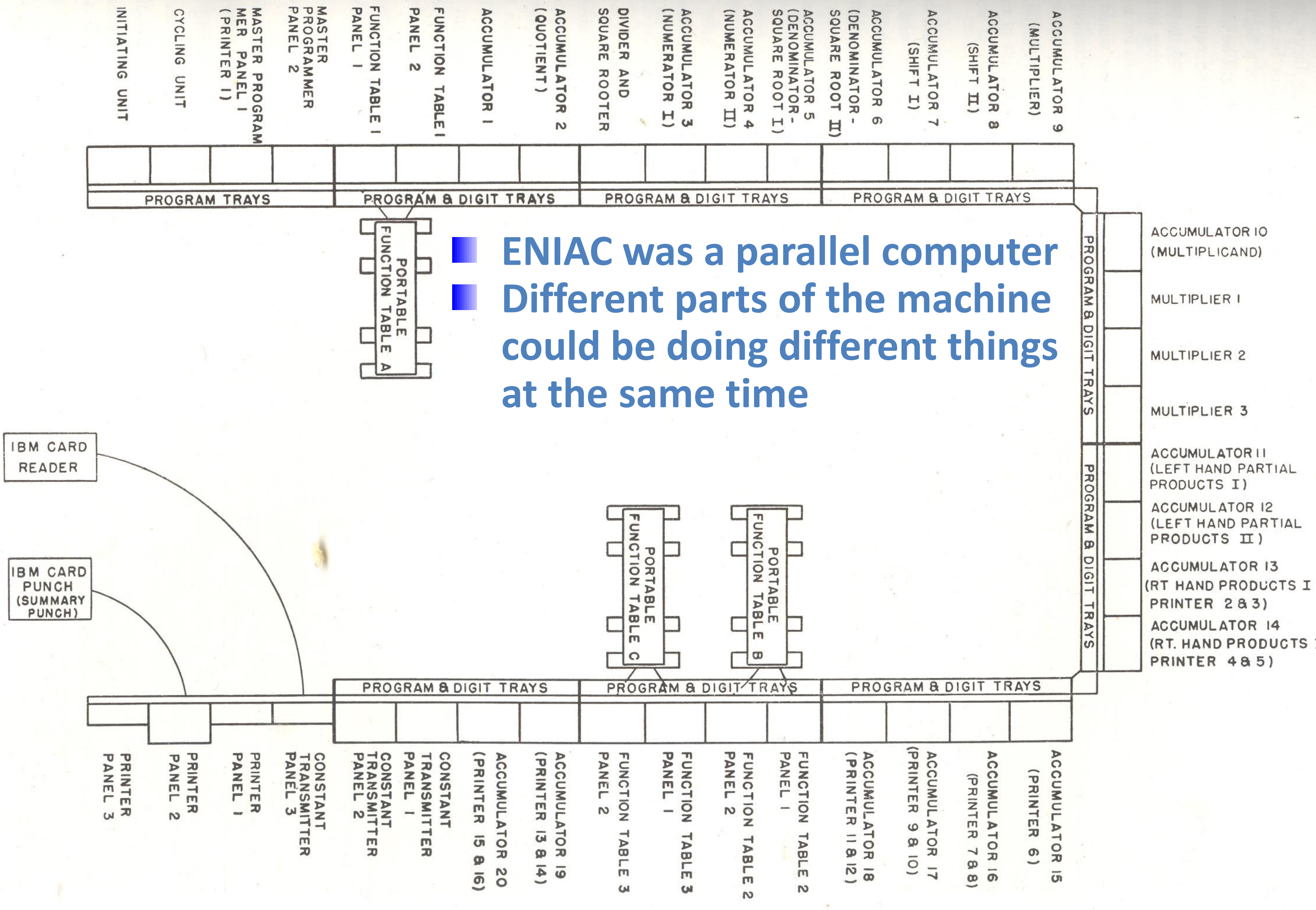
If in doubt, read the guidelines available at the link above ☺

J Presper Eckert (1919-1995)



Co-inventor of, and chief engineer on, the ENIAC, arguably the first general-purpose computer (first operational Feb 14th 1946)

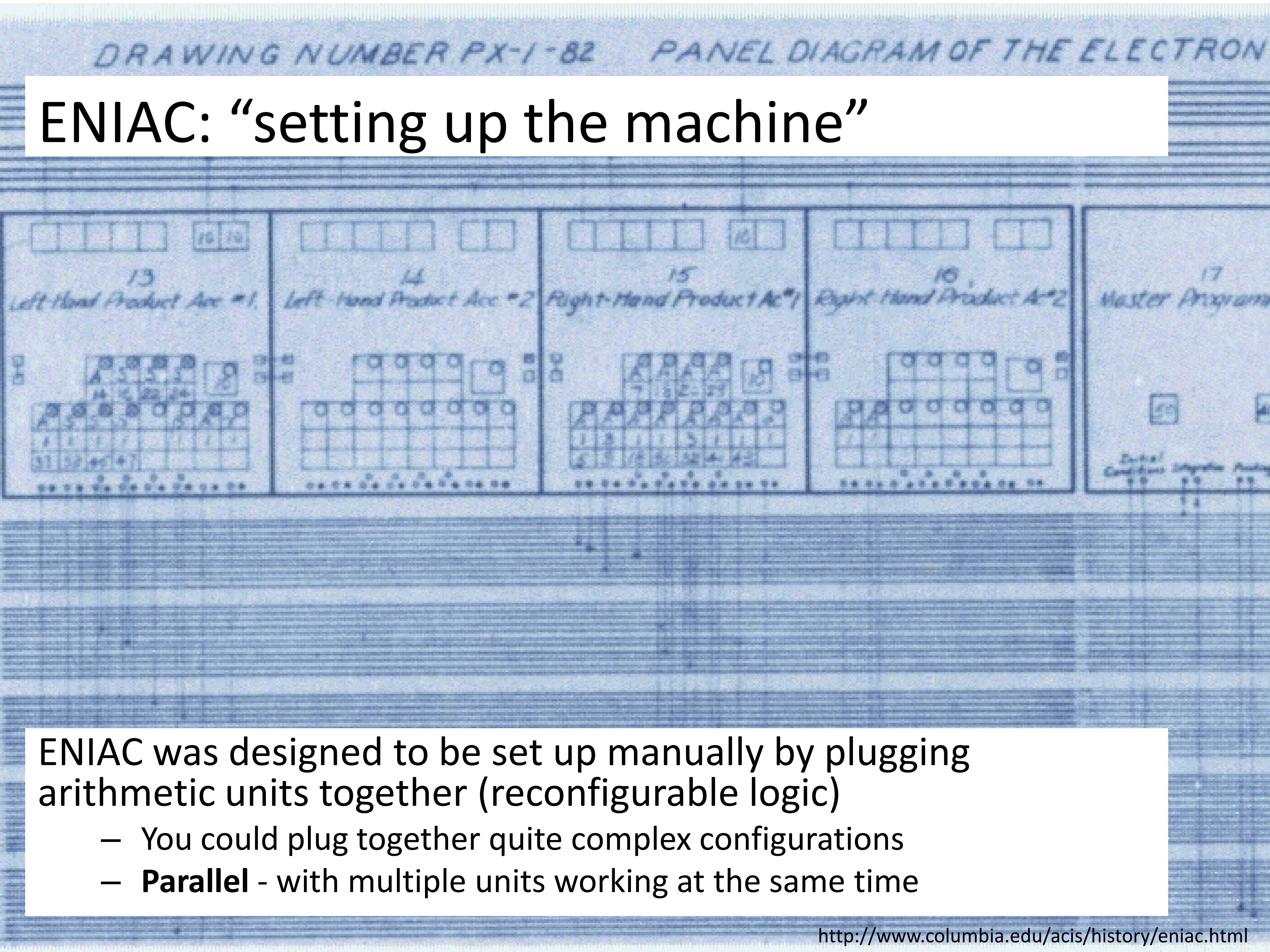
27 tonnes, 150KW, 5000 cycles/sec


INITIATING UNIT
CYCLING UNIT
MASTER PROGRAMMER PANEL 1 (PRINTER 1)
MASTER PROGRAMMER PANEL 2
FUNCTION TABLE 1 PANEL 1
FUNCTION TABLE 1 PANEL 2
ACCUMULATOR 1
ACCUMULATOR 2 (QUOTIENT)
DIVIDER AND SQUARE ROOTER
ACCUMULATOR 3 (NUMERATOR I)
ACCUMULATOR 4 (NUMERATOR II)
ACCUMULATOR 5 (DENOMINATOR-SQUARE ROOT I)
ACCUMULATOR 6 (DENOMINATOR-SQUARE ROOT II)
ACCUMULATOR 7 (SHIFT I)
ACCUMULATOR 8 (SHIFT II)
ACCUMULATOR 9 (MULTIPLIER)
PROGRAM TRAYS
PROGRAM & DIGIT TRAYS
PROGRAM & DIGIT TRAYS
PROGRAM & DIGIT TRAYS
PORTABLE FUNCTION TABLE A
PROGRAM & DIGIT TRAYS
PROGRAM & DIGIT TRAYS
ACCUMULATOR 10 (MULTIPLICAND)
MULTIPLIER 1
MULTIPLIER 2
MULTIPLIER 3
ACCUMULATOR 11 (LEFT HAND PARTIAL PRODUCTS I)
ACCUMULATOR 12 (LEFT HAND PARTIAL PRODUCTS II)
ACCUMULATOR 13 (RT HAND PRODUCTS I PRINTER 2 & 3)
ACCUMULATOR 14 (RT. HAND PRODUCTS PRINTER 4 & 5)
IBM CARD READER
IBM CARD PUNCH (SUMMARY PUNCH)
PORTABLE FUNCTION TABLE C
PORTABLE FUNCTION TABLE B
PROGRAM & DIGIT TRAYS
PROGRAM & DIGIT TRAYS
PROGRAM & DIGIT TRAYS
PRINTER PANEL 3
PRINTER PANEL 2
PRINTER PANEL 1
CONSTANT TRANSMITTER PANEL 3
CONSTANT TRANSMITTER PANEL 2
CONSTANT TRANSMITTER PANEL 1
ACCUMULATOR 20 (PRINTER 15 & 16)
ACCUMULATOR 19 (PRINTER 13 & 14)
FUNCTION TABLE 3 PANEL 2
FUNCTION TABLE 3 PANEL 1
FUNCTION TABLE 2 PANEL 2
FUNCTION TABLE 2 PANEL 1
ACCUMULATOR 18 (PRINTER 11 & 12)
ACCUMULATOR 17 (PRINTER 9 & 10)
ACCUMULATOR 16 (PRINTER 7 & 8)
ACCUMULATOR 15 (PRINTER 6)

- ENIAC was a parallel computer
- Different parts of the machine could be doing different things at the same time

J.G. Brainerd & T.K. Sharpless. "The ENIAC." pp 163-172 Electrical Engineering, Feb 1948.

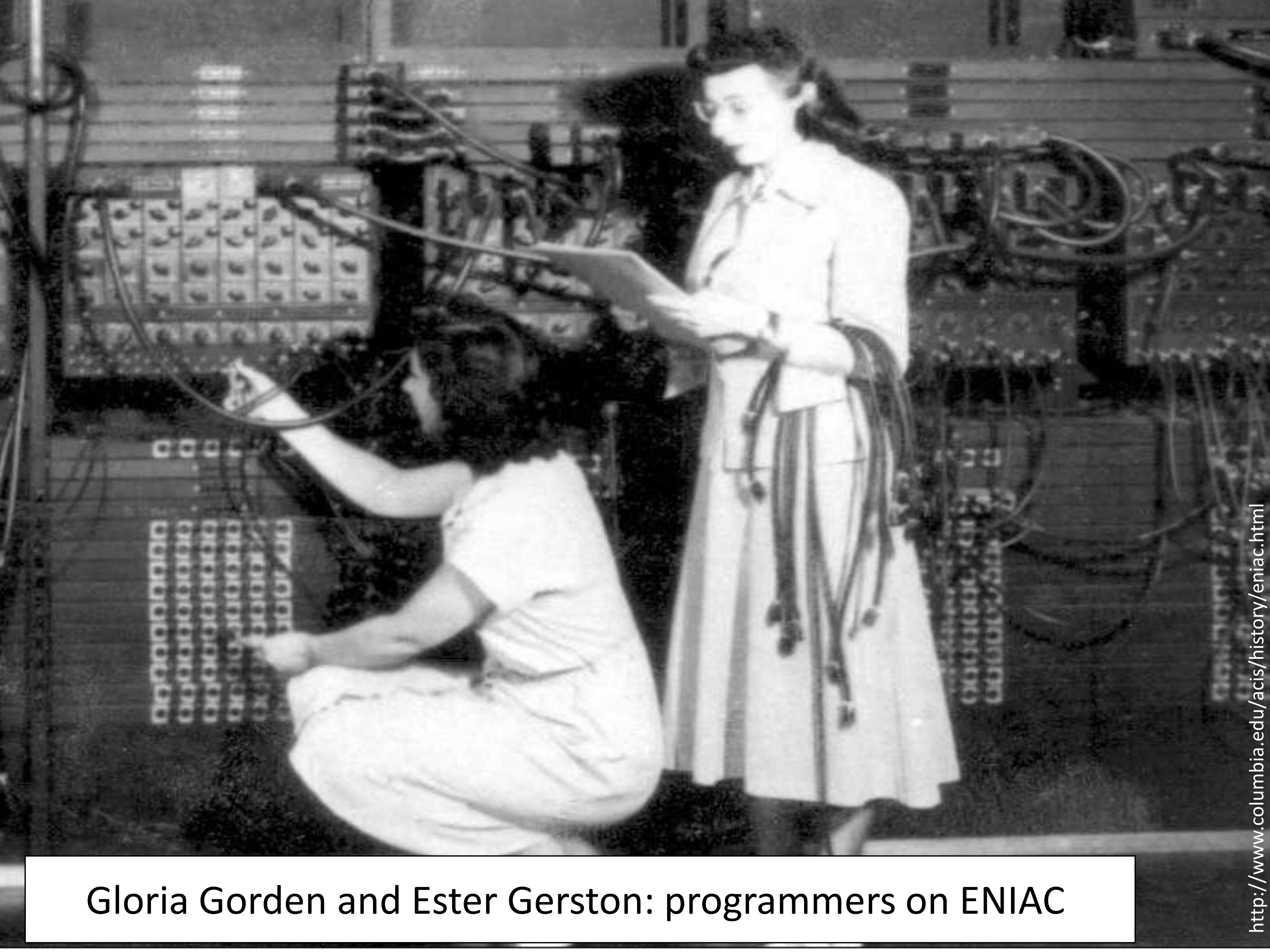
ENIAC: “setting up the machine”



ENIAC was designed to be set up manually by plugging arithmetic units together (reconfigurable logic)

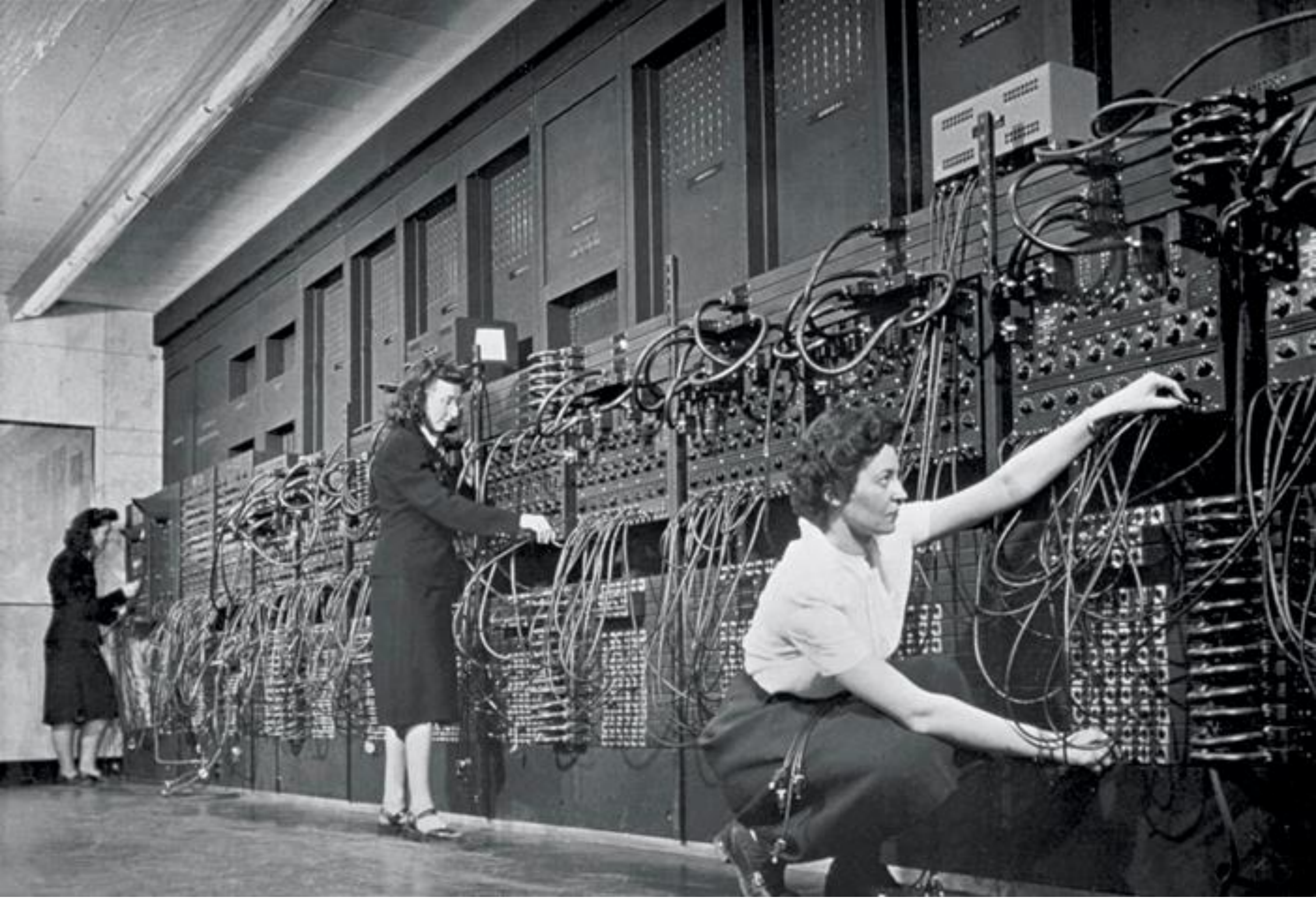
- You could plug together quite complex configurations
- **Parallel** - with multiple units working at the same time

http://www.columbia.edu/acis/history/eniac.html



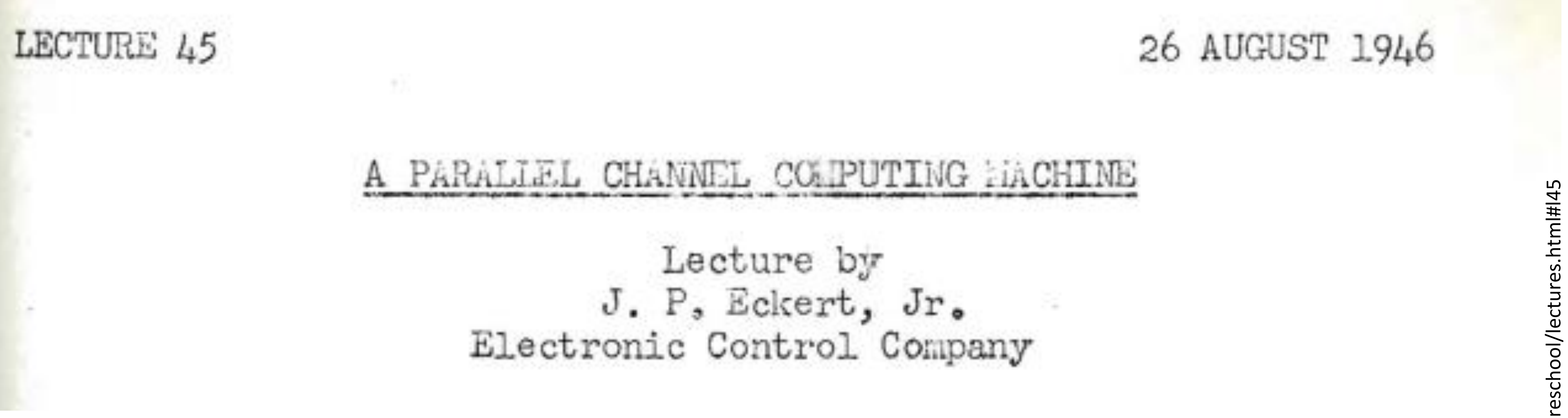
Gloria Gorden and Ester Gerston: programmers on ENIAC

http://www.columbia.edu/acis/history/eniac.html



Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC

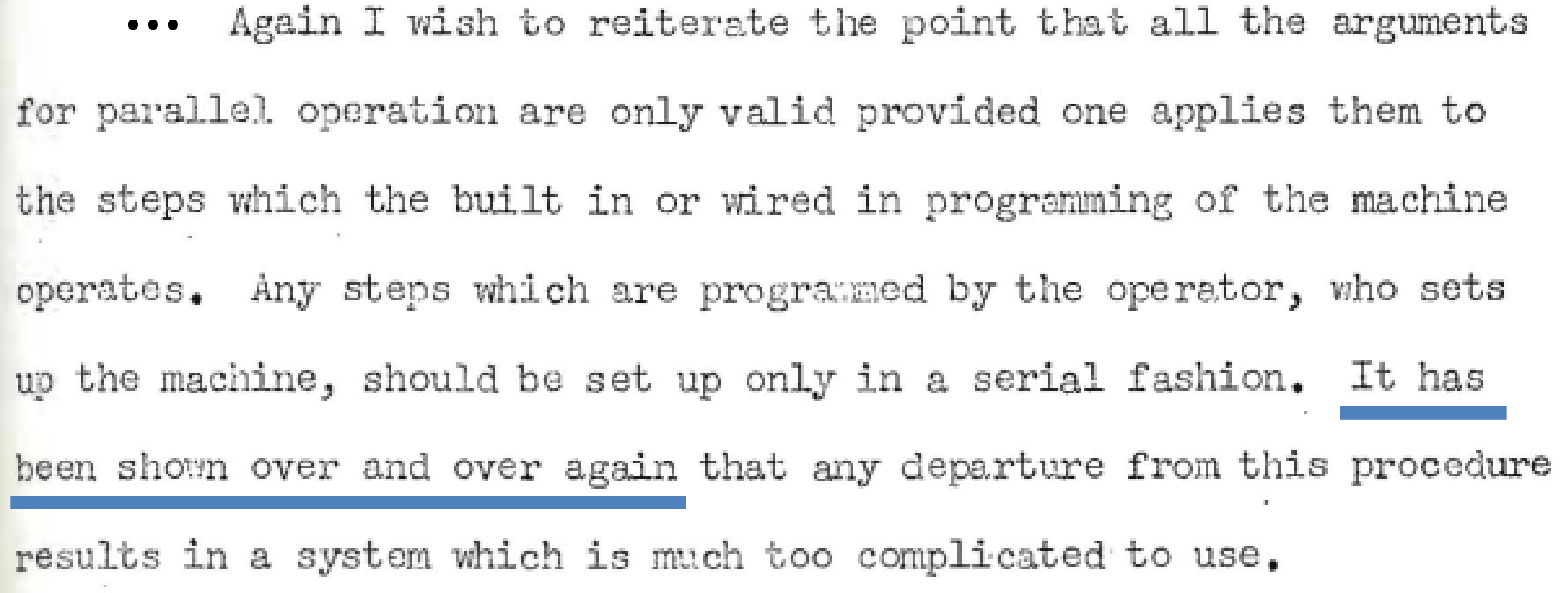
https://imgur.com/gallery/nh38c and http://fortune.com/2014/09/18/walter-isaacson-the-women-of-eniac/



LECTURE 45 26 AUGUST 1946

A PARALLEL CHANNEL COMPUTING MACHINE

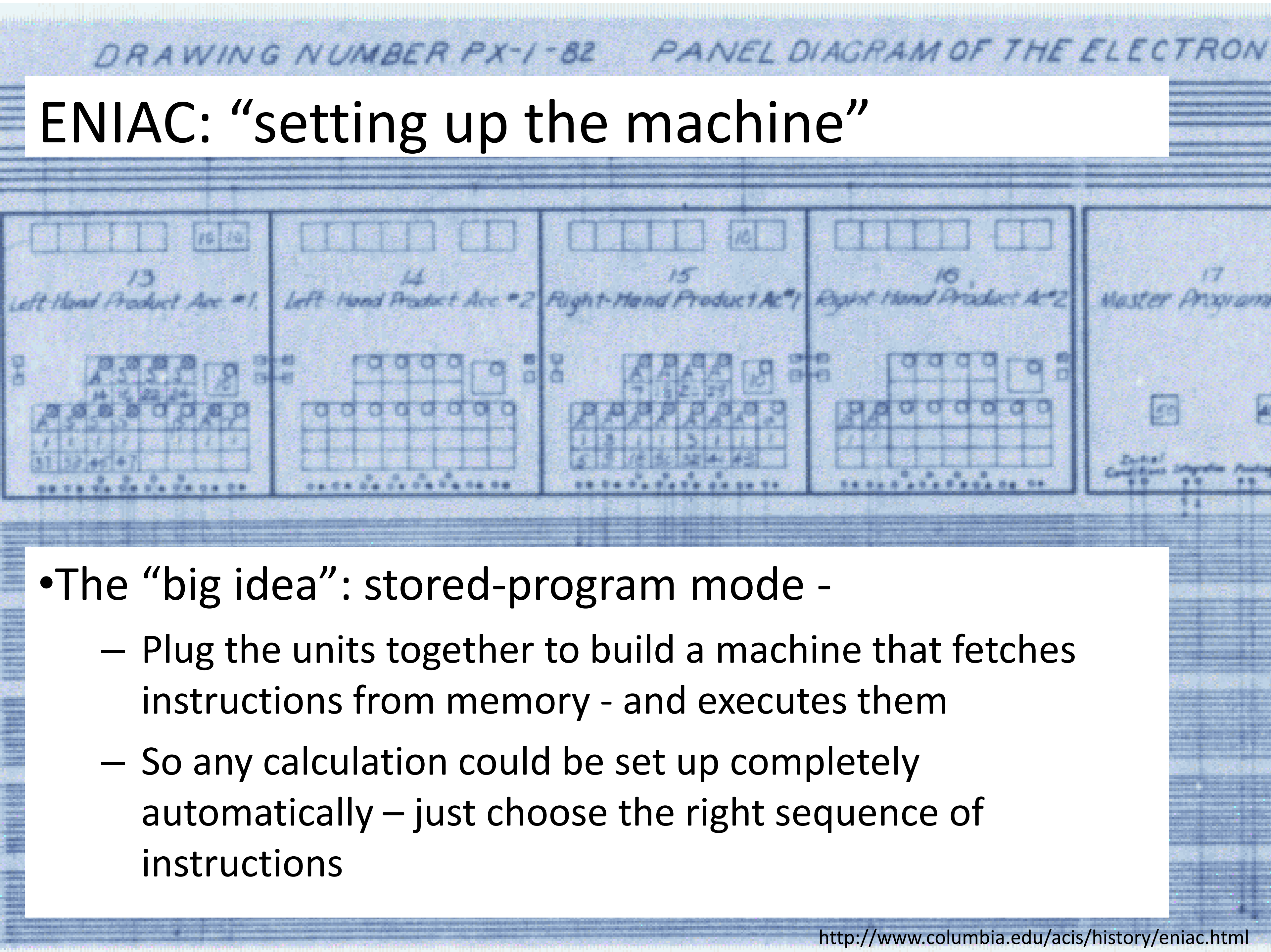
Lecture by
J. P, Eckert, Jr.
Electronic Control Company



... Again I wish to reiterate the point that all the arguments for parallel operation are only valid provided one applies them to the steps which the built in or wired in programming of the machine operates. Any steps which are programmed by the operator, who sets up the machine, should be set up only in a serial fashion. It has been shown over and over again that any departure from this procedure results in a system which is much too complicated to use.

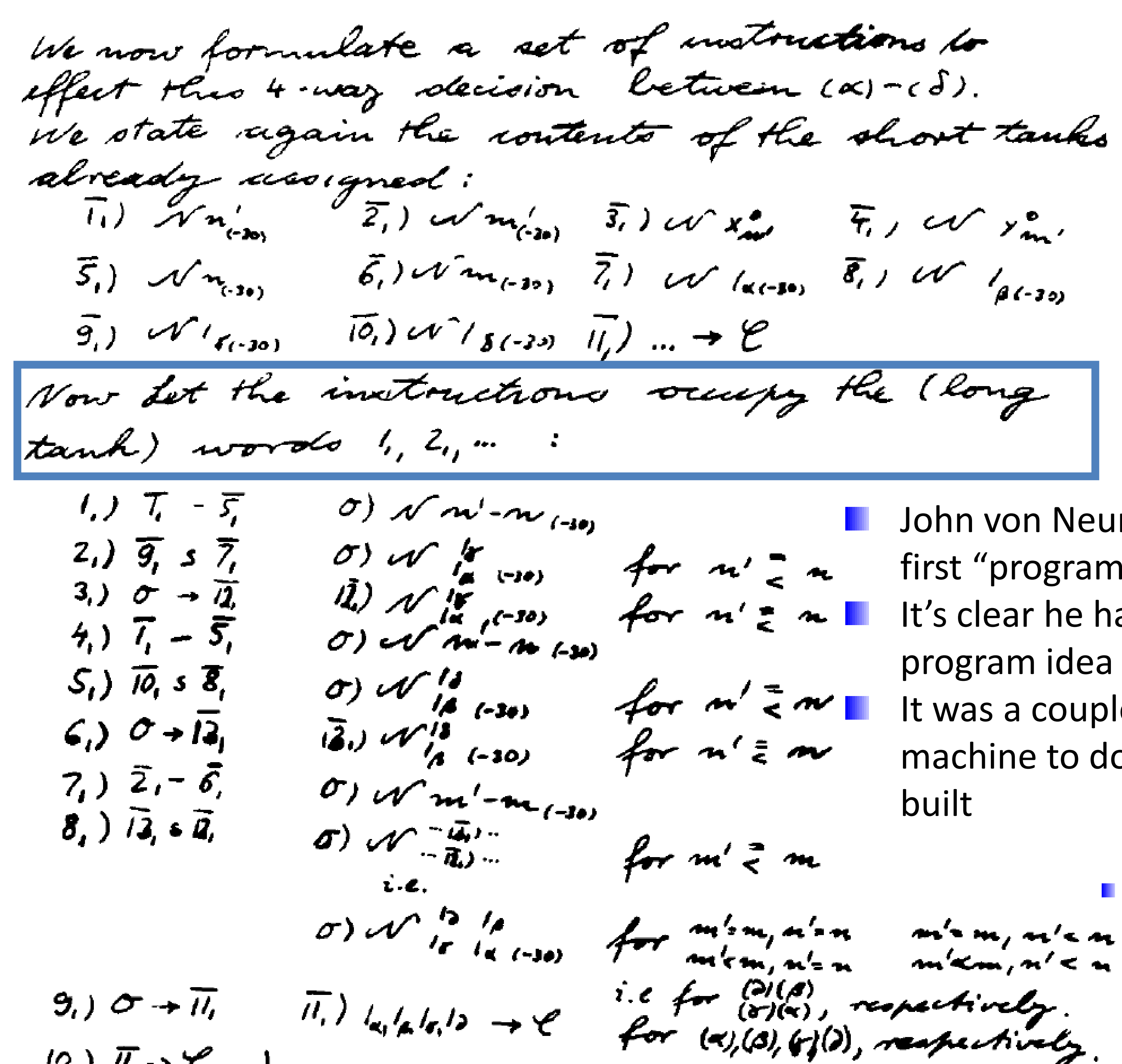
See also http://www.digital60.org/birth/themooreschool/lectures.html#l45

ENIAC: “setting up the machine”



- The “big idea”: stored-program mode -
  - Plug the units together to build a machine that fetches instructions from memory - and executes them
  - So any calculation could be set up completely automatically – just choose the right sequence of instructions

http://www.columbia.edu/acis/history/eniac.html



We now formulate a set of instructions to effect this 4-way decision between (α)–(δ).
We state again the contents of the short tanks already assigned:

$\bar{1}_1$) $N\ n'_{(-30)}$ $\bar{2}_1$) $N\ m'_{(-30)}$ $\bar{3}_1$) $N\ x^0_{m}$ $\bar{4}_1$) $N\ y^0_{m'}$

$\bar{5}_1$) $N\ n_{(-30)}$ $\bar{6}_1$) $N\ m_{(-30)}$ $\bar{7}_1$) $N\ 1_{\alpha(-30)}$ $\bar{8}_1$) $N\ 1_{\beta(-30)}$

$\bar{9}_1$) $N\ 1_{\gamma(-30)}$ $\overline{10}_1$) $N\ 1_{\delta(-30)}$ $\overline{11}_1$) … → $\mathcal{C}$

Now let the instructions occupy the (long tank) words 1, 2, … :

1,) $\bar{1}_1 - \bar{5}_1$ σ) $N\ n'-n_{(-30)}$

2,) $\bar{9}_1\ s\ \bar{7}_1$ σ) $N\ {}^{1_\gamma}_{1_\alpha}\ (-30)$ for $n' \overset{=}{<} n$

3,) $\sigma \to \overline{12}_1$ $\overline{12}_1$) $N\ {}^{1_\gamma}_{1_\alpha}\ (-30)$ for $n' \overset{=}{<} n$

4,) $\bar{1}_1 - \bar{5}_1$ σ) $N\ m'-m_{(-30)}$

5,) $\overline{10}_1\ s\ \bar{8}_1$ σ) $N\ {}^{1_\delta}_{1_\beta}\ (-30)$ for $m' \overset{=}{<} m$

6,) $\sigma \to \overline{13}_1$ $\overline{13}_1$) $N\ {}^{1_\delta}_{1_\beta}\ (-30)$ for $m' \overset{=}{<} m$

7,) $\bar{2}_1 - \bar{6}_1$ σ) $N\ m'-m_{(-30)}$

8,) $\overline{13}_1\ s\ \overline{12}_1$ σ) $N\ {}^{\dots \overline{13}_1) \dots}_{\dots \overline{12}_1) \dots}$ for $m' \overset{=}{<} m$

i.e.

σ) $N\ {}^{1_\delta}_{1_\gamma}\ {}^{1_\beta}_{1_\alpha}\ (-30)$ for $\begin{matrix} m'=m,\ n'=n & m'=m,\ n'<n \\ m'<m,\ n'=n & m'<m,\ n'<n \end{matrix}$

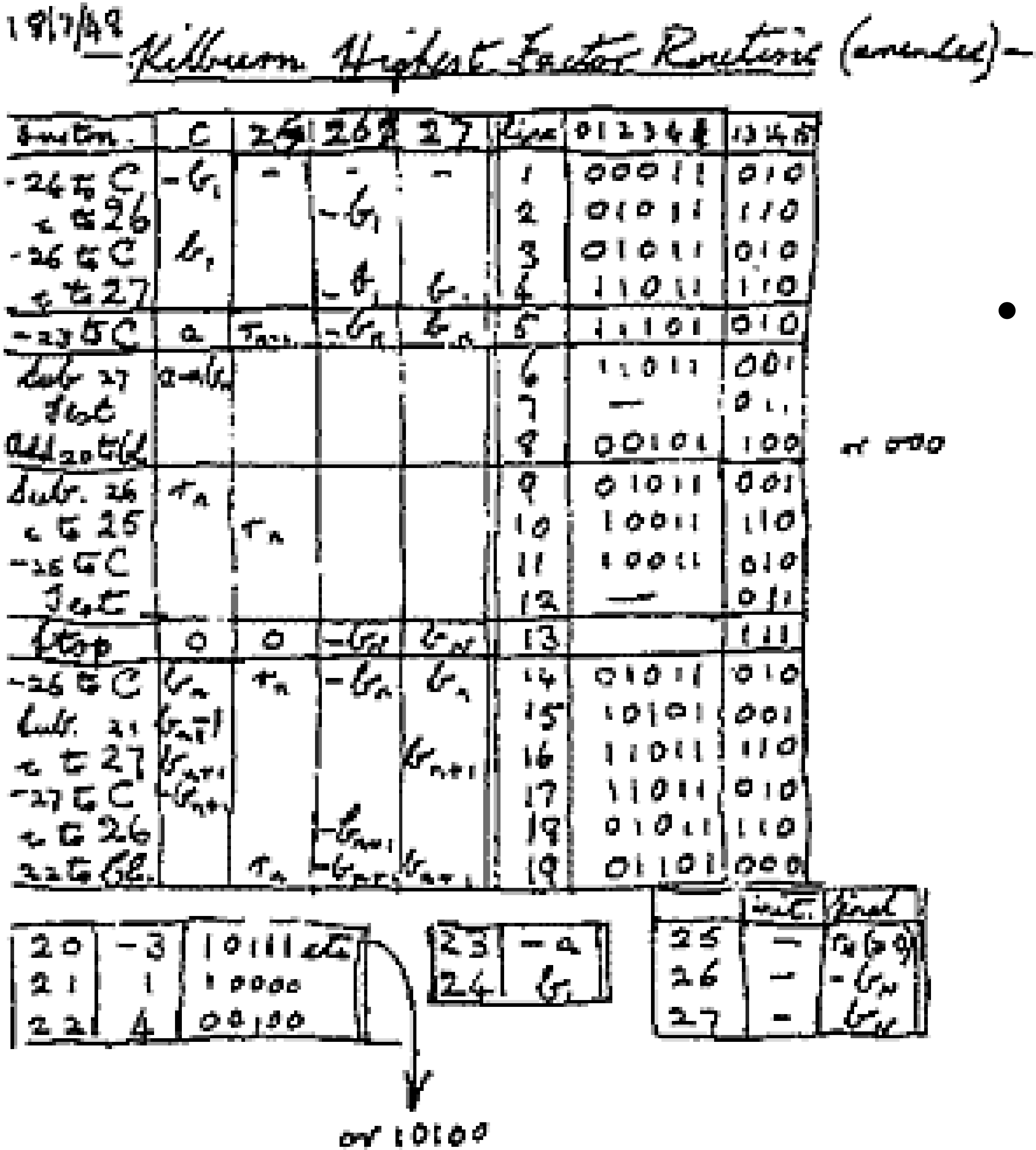
i.e for $\begin{matrix}(\delta)(\beta)\\(\gamma)(\alpha)\end{matrix}$, respectively.

9,) $\sigma \to \overline{11}_1$ $\overline{11}_1$) $1_\alpha, 1_\beta, 1_\gamma, 1_\delta \to \mathcal{C}$ for (α), (β), (γ), (δ), respectively.

10,) $\overline{11} \to \mathcal{C}$

- John von Neumann wrote his first “program” in 1945
- It’s clear he had the stored program idea in mind
- It was a couple of years before a machine to do it was actually built

- Knuth, D. E. 1970. Von Neumann's First Computer Program. ACM Comput. Surv. 2, 4 (Dec. 1970), 247-260.



18/7/48 — Kilburn Highest Factor Routine (amended) —

Instruction.	C	25	26	27	Line	01234	13 14 15
-24 to C	$-b_1$	–	–	–	1	00011	010
c to 26			$-b_1$		2	01011	110
-26 to C	b_1				3	01011	010
c to 27			$-b_1$	b_1	4	11011	110
-23 to C	a	r_{n-1}	$-b_n$	b_n	5	11101	010
Sub 27	$a-b_n$				6	11011	001
Test					7	—	011
Add 20 to Cl					8	00101	100
Sub. 26	r_n				9	01011	001
c to 25		r_n			10	10011	110
-25 to C					11	10011	010
Test					12	—	011
Stop	0	0	$-b_N$	b_N	13		111
-26 to C	b_n	r_n	$-b_n$	b_n	14	01011	010
Sub. 21	$b_n - 1$				15	10101	001
c to 27	b_{n+1}			b_{n+1}	16	11011	110
-27 to C	$-b_{n+1}$				17	11011	010
c to 26			$-b_{n+1}$		18	01011	110
22 to Cl.		r_n	$-b_{n+1}$	b_{n+1}	19	01101	000

or 000

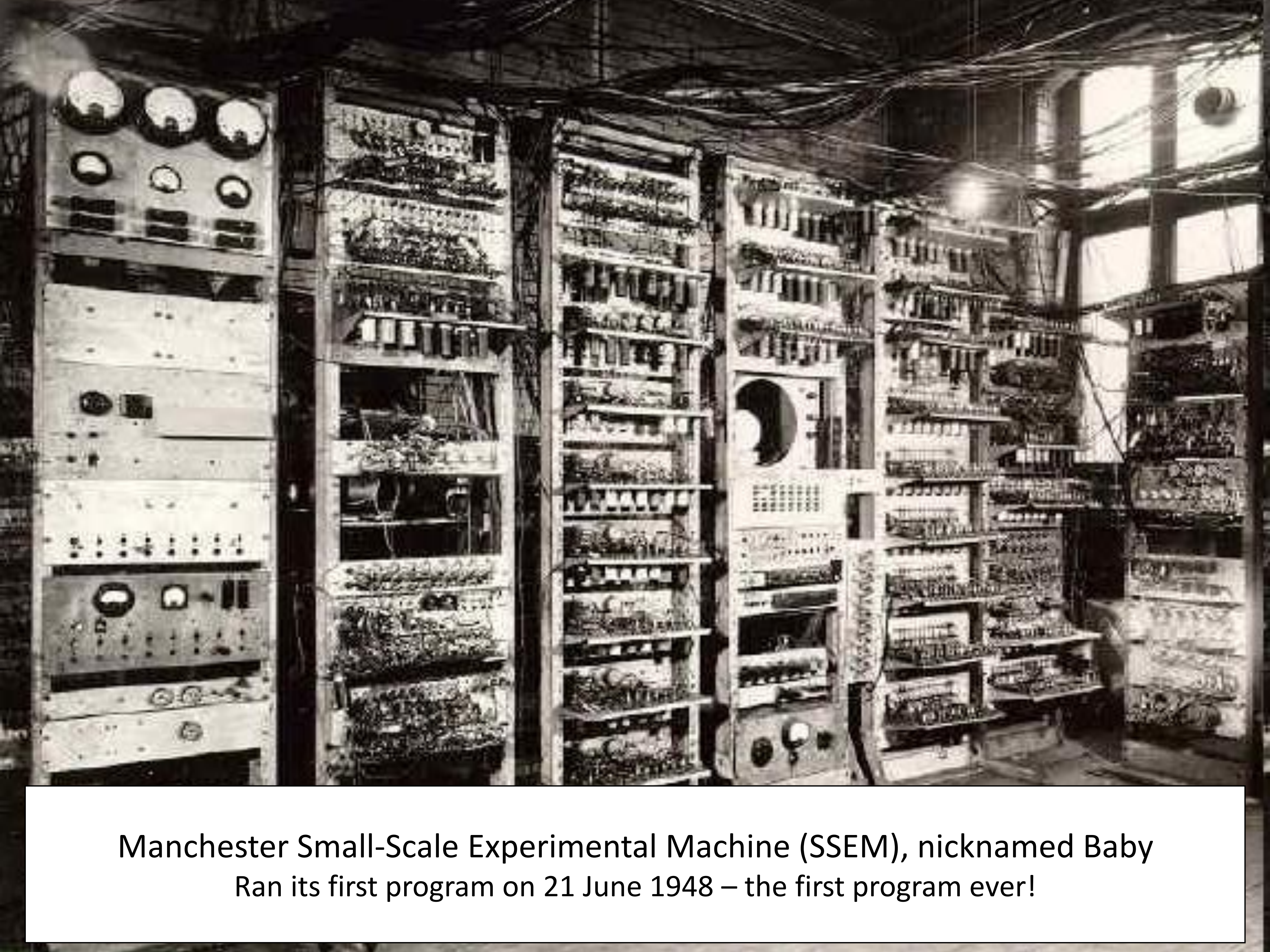
20	-3	10111 etc.
21	1	10000
22	4	00100

or 10100

23	-a
24	b_1

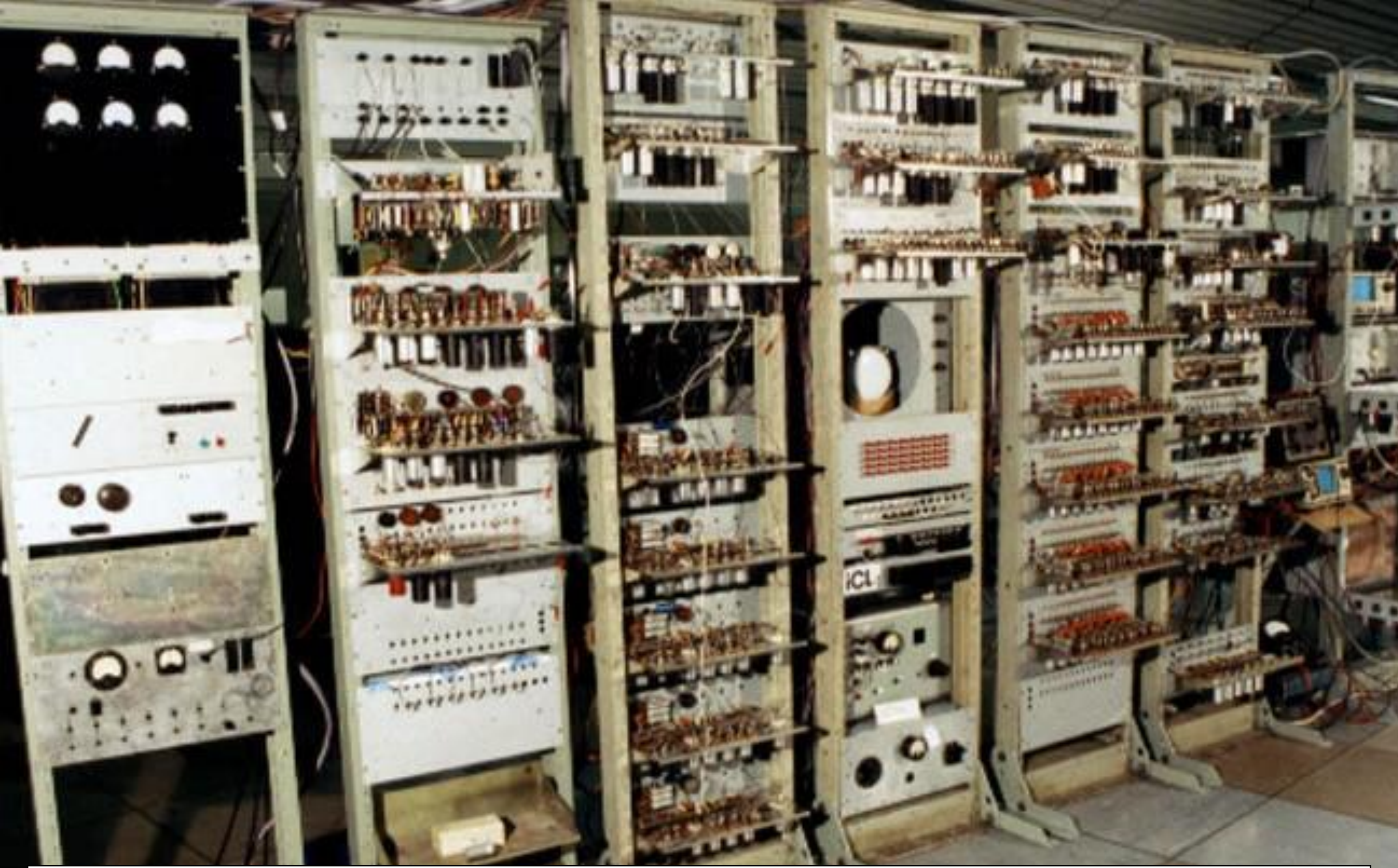
	init.	final
25	—	r_n(0)
26	—	$-b_N$
27	—	b_N

- This is the first program to actually run!



Manchester Small-Scale Experimental Machine (SSEM), nicknamed Baby

Ran its first program on 21 June 1948 – the first program ever!



Manchester Small-Scale Experimental Machine (SSEM), nicknamed Baby

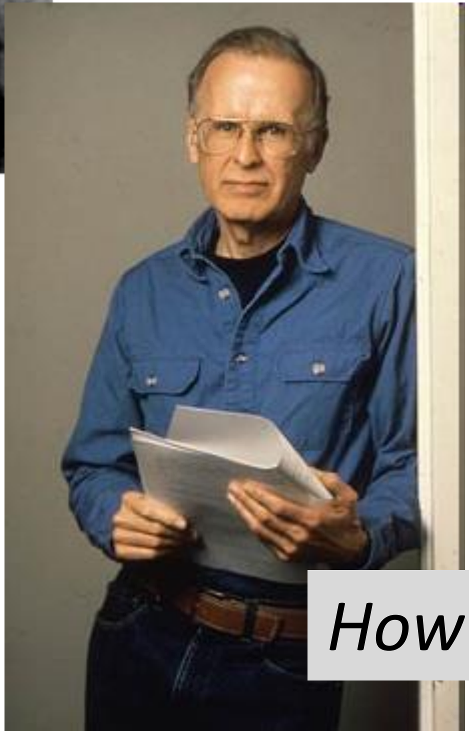
Rebuilt for the 60th anniversary, now in in the Museum of Science and Industry in Manchester

The “von Neumann bottleneck”

Wikipedia, http://www.lanl.gov/history/atomicbomb/images/NeumannL.GIF

John von Neumann
http://en.wikipedia.org/wiki/John_von_Neumann

John Backus
“**Can Programming be Liberated from the von Neumann Style?”** (1979)



www.post-gazette.com/pg/07080/771123-96.stm

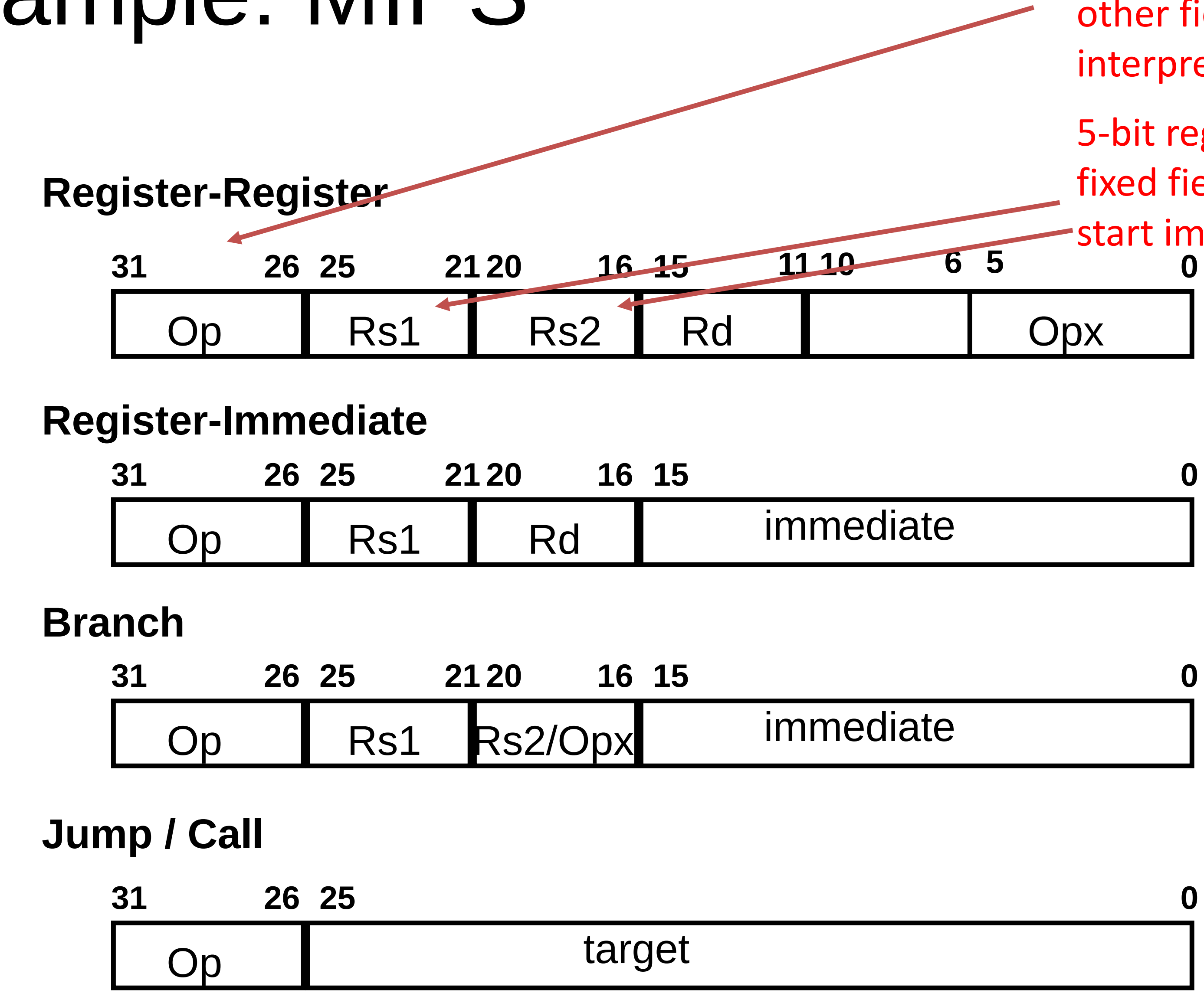
The price to pay:

- **Stored-program mode was serial – one instruction at a time**

- How can we have our cake - and eat it?
  - **Flexibility and ease of programming**
  - **Performance of parallelism**

How to beat the “Turing Tax”

Example: MIPS



Opcode specifies how other fields will be interpreted

5-bit register specifier at fixed field so access can start immediately

Register-Register

31 26	25 21	20 16	15 11	10 6	5 0
Op	Rs1	Rs2	Rd		Opx

Register-Immediate

31 26	25 21	20 16	15 0
Op	Rs1	Rd	immediate

Branch

31 26	25 21	20 16	15 0
Op	Rs1	Rs2/Opx	immediate

Jump / Call

31 26	25 0
Op	target

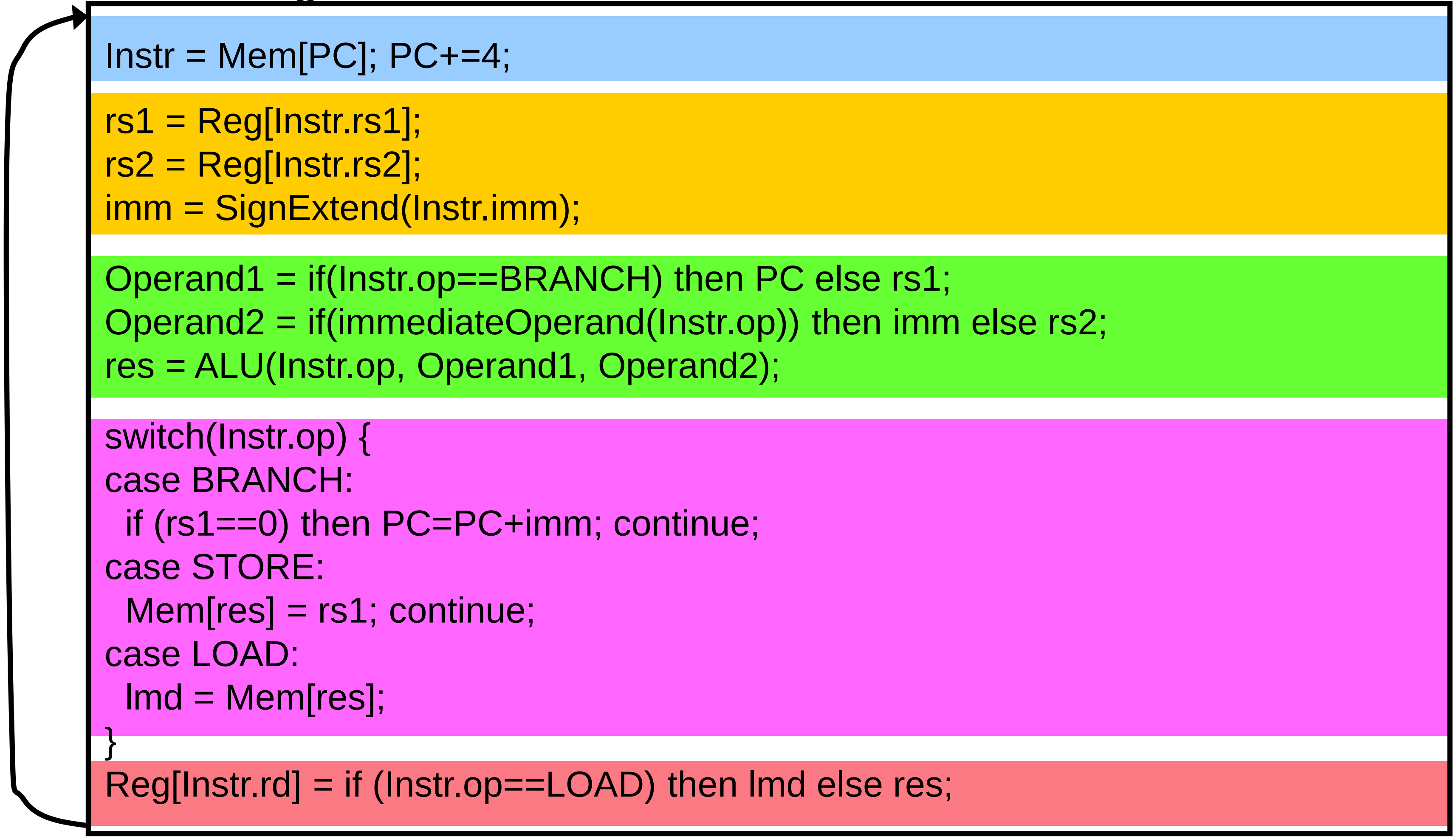
Q: How many registers can we address?

Q: What is the largest signed immediate operand for “subw r1,r2,X”?

Q: What range of addresses can a conditional branch jump to?

A machine to execute these instructions

- To execute this instruction set we need a machine that fetches them and does what each instruction says
- A “universal” computing device – a simple digital circuit that, with the right code, can compute *anything*
- Something like:



```
Instr = Mem[PC]; PC+=4;

rs1 = Reg[Instr.rs1];
rs2 = Reg[Instr.rs2];
imm = SignExtend(Instr.imm);

Operand1 = if(Instr.op==BRANCH) then PC else rs1;
Operand2 = if(immediateOperand(Instr.op)) then imm else rs2;
res = ALU(Instr.op, Operand1, Operand2);

switch(Instr.op) {
case BRANCH:
  if (rs1==0) then PC=PC+imm; continue;
case STORE:
  Mem[res] = rs1; continue;
case LOAD:
  lmd = Mem[res];
}
Reg[Instr.rd] = if (Instr.op==LOAD) then lmd else res;
```

5 Steps of MIPS Datapath

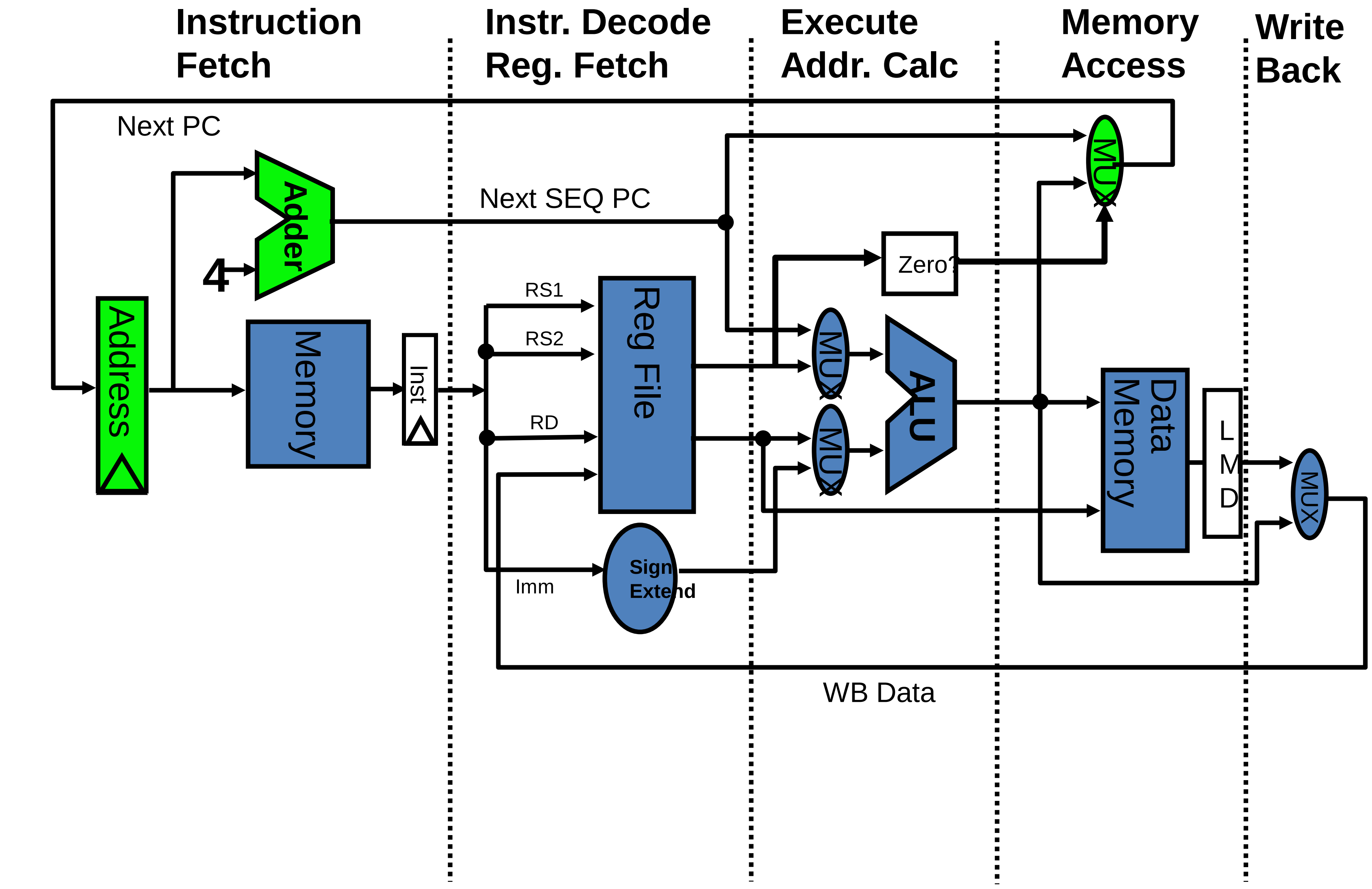


Figure 3.1, Page 130, CA:AQA 2e

5-stage MIPS pipeline with pipeline buffers

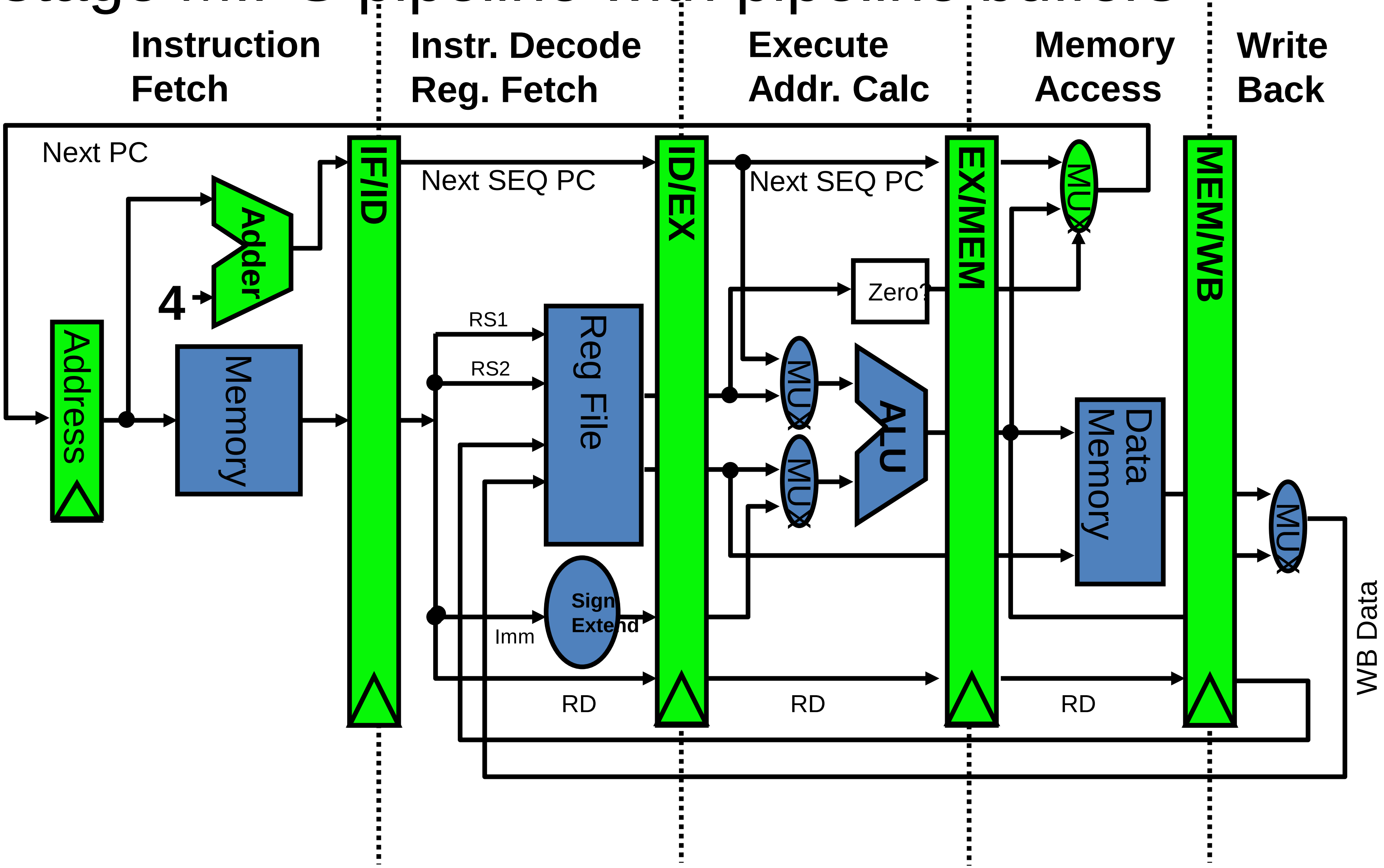
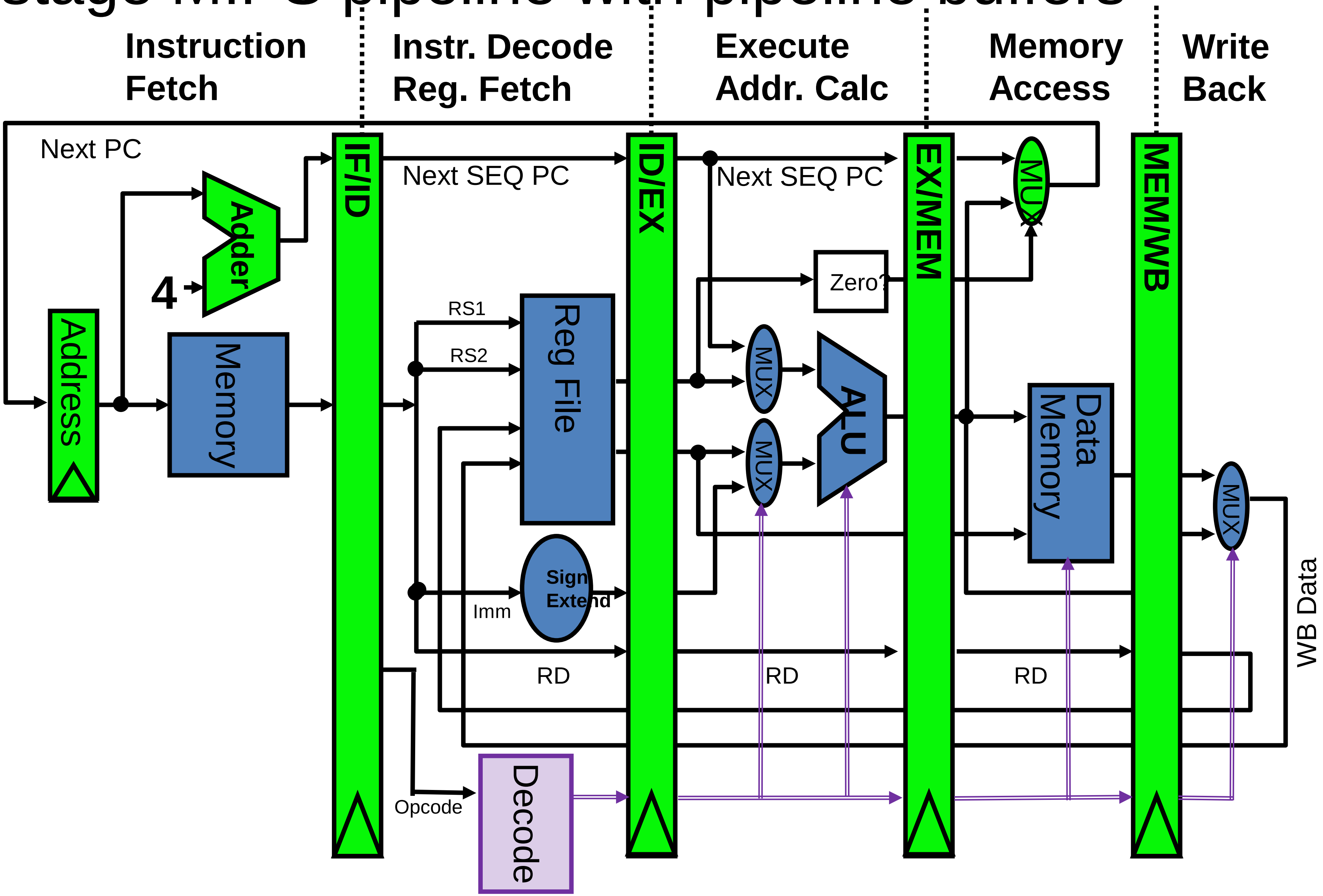


Figure 3.4, Page 134 , CA:AQA 2e

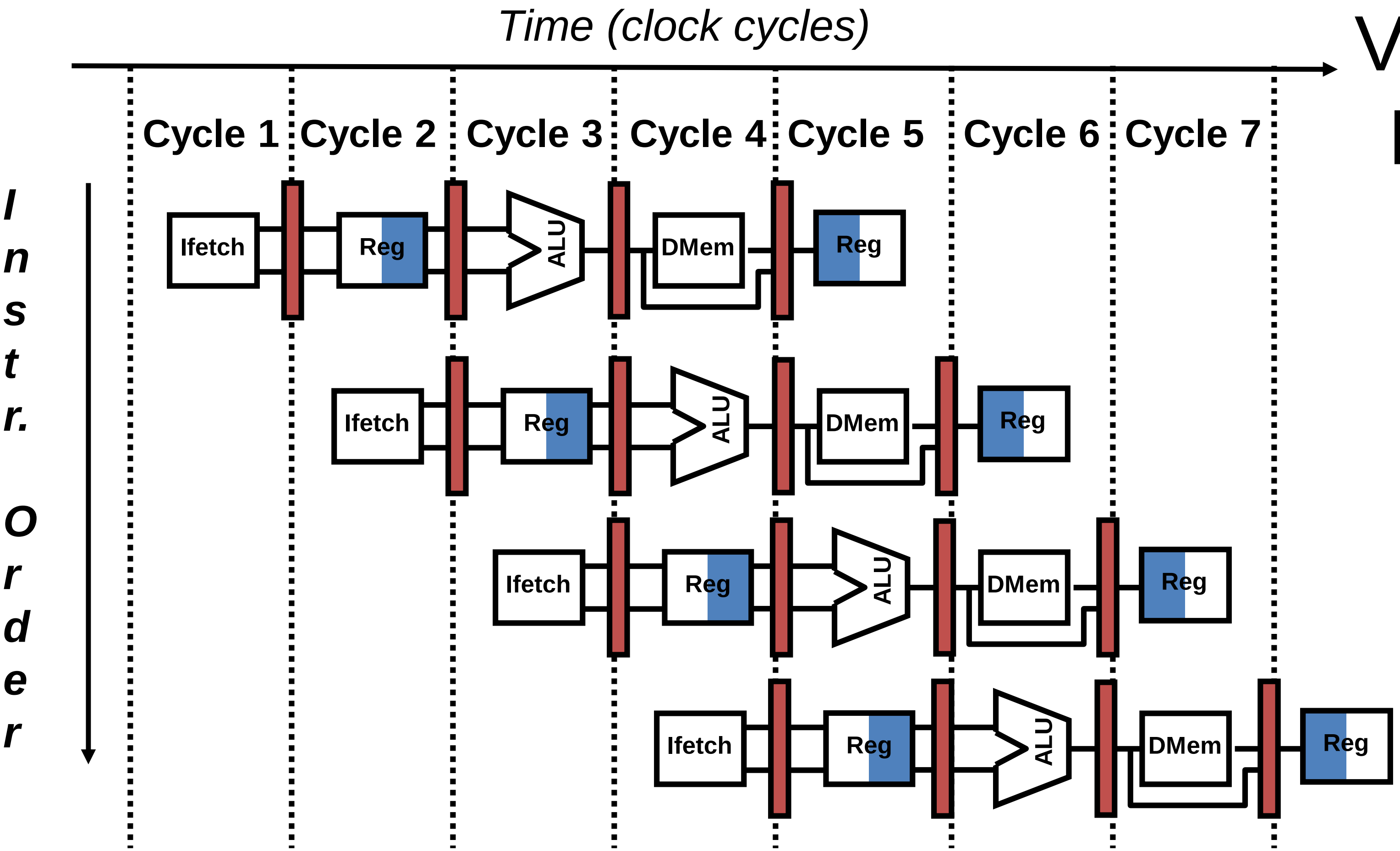
5-stage MIPS pipeline with pipeline buffers



- **Data stationary control**
  - **Control signals are needed to configure the MUXes, ALU, read/write**
  - **Carried with the corresponding instruction along the pipeline**

Figure 3.4, Page 134 , CA:AQA 2e

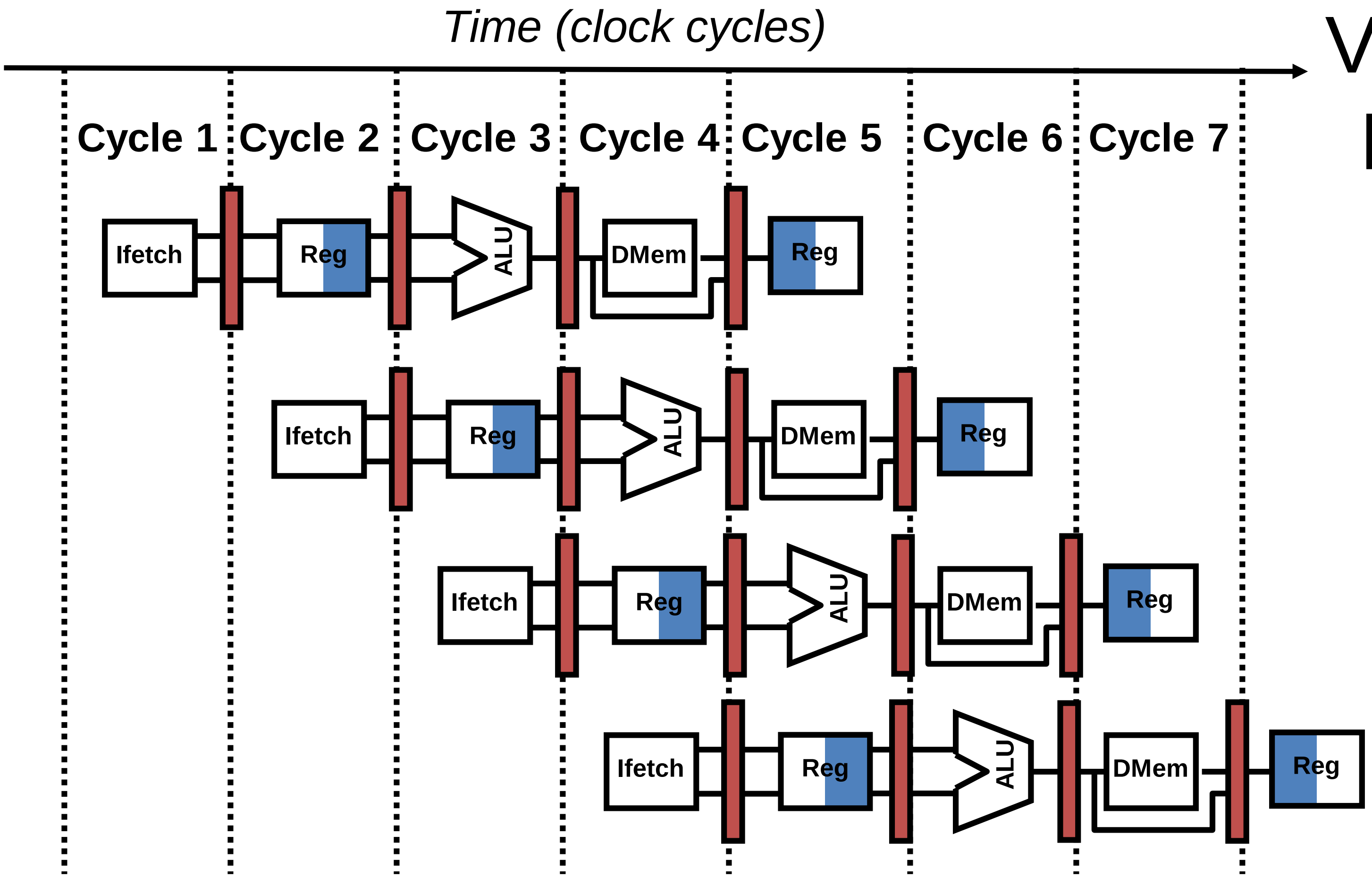
Visualizing Pipelining



- Pipelining doesn't help **latency** of single instruction
  - it helps **throughput** of entire workload
- Pipeline rate limited by **slowest** pipeline stage
- Potential speedup = **Number pipe stages**
- Unbalanced lengths of pipe stages reduces speedup
- Time to "**fill**" pipeline and time to "**drain**" it reduces speedup
- Speedup comes from parallelism - for free – no new hardware
- **Many pipelines are more complex - Pentium 4 "Netburst" has 31 stages.**

Figure 3.3, Page 133 , CA:AQA 2e

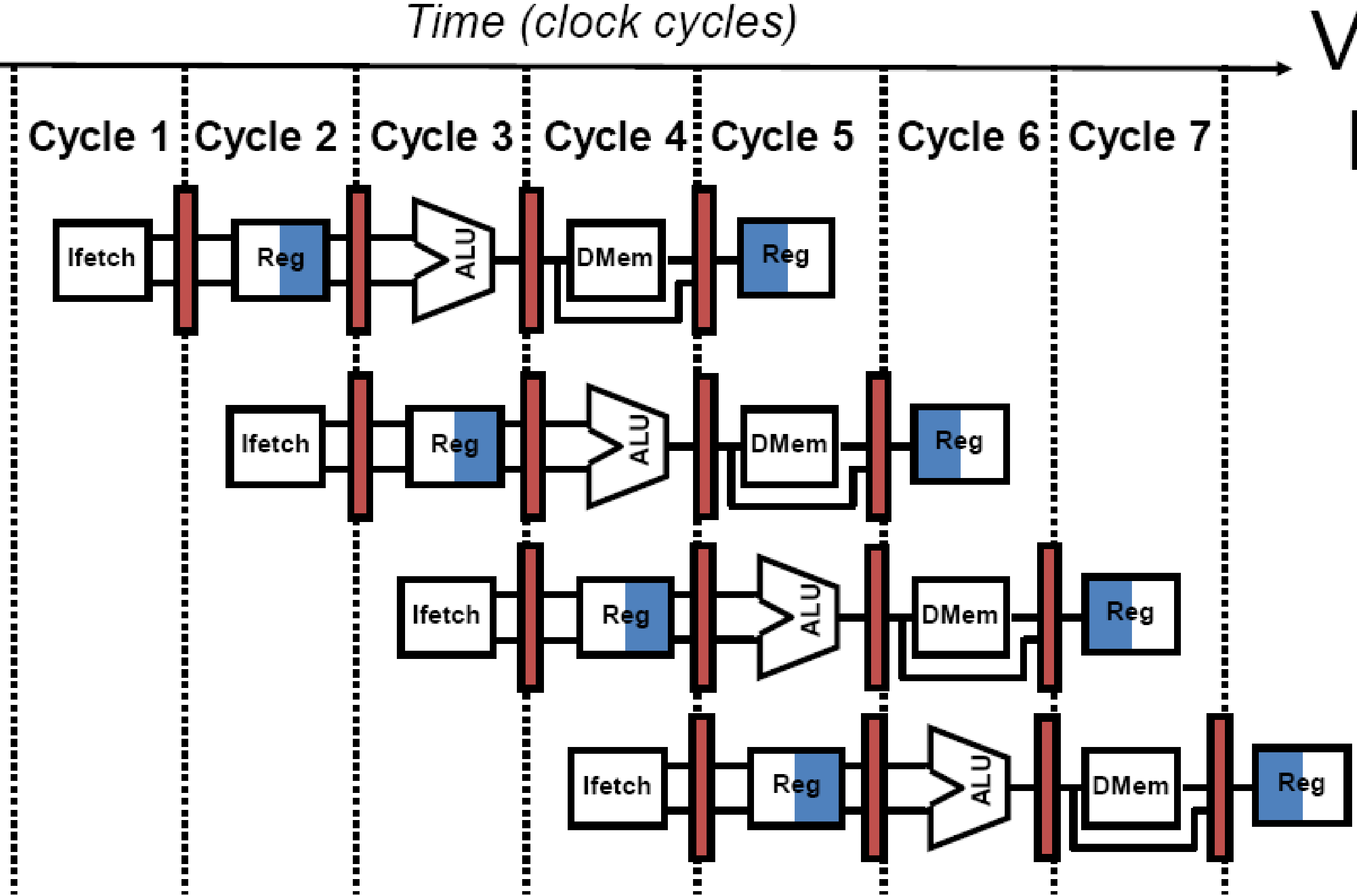
Visualizing Pipelining



- Pipelining doesn’t help latency of single instruction
  - it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to “fill” pipeline and time to “drain” it reduces speedup
- Speedup comes from parallelism - for free – no new hardware
- Many pipelines are more complex - Pentium 4 “Netburst” has 31 stages.

Figure 3.3, Page 133 , CA:AQA 2e

Vis
P



pelining doesn’t help latency of single instruction

- it helps throughput of entire workload

peline rate limited by slowest pipeline stage

tential speedup = Number pipe stages

balanced lengths of pipe stages reduces speedup

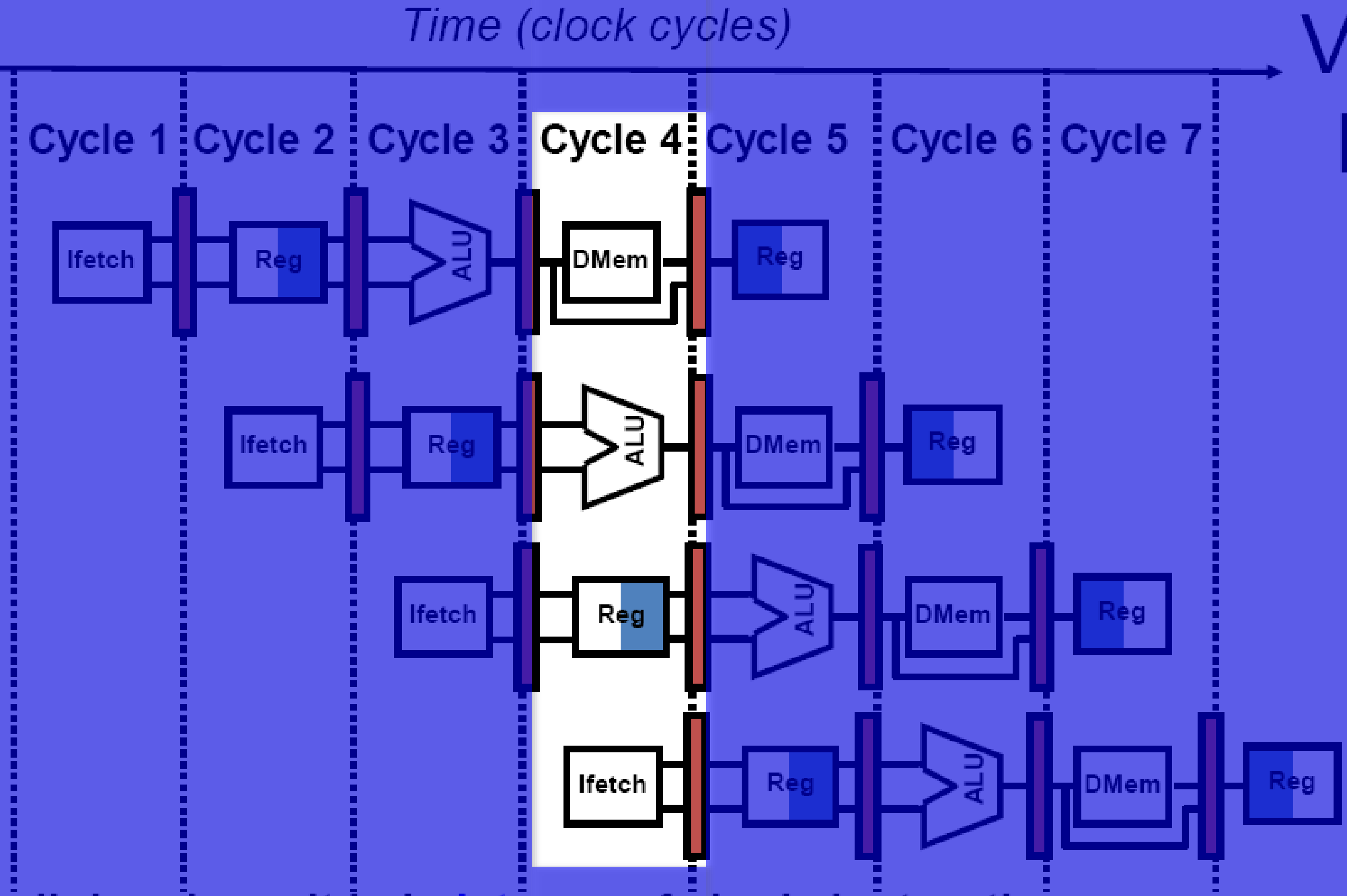
ne to “fill” pipeline and time to “drain” it reduces speedup

eedup comes from parallelism - for free – no new hardware

ny pipelines are more complex - Pentium 4 “Netburst” has 31 s

ge 133 , CA:AQA 2e

Vis
P



pelining doesn't help latency of single instruction

it helps throughput of entire workload

peline rate limited by slowest pipeline stage

tential speedup = Number pipe stages

balanced lengths of pipe stages reduces speedup

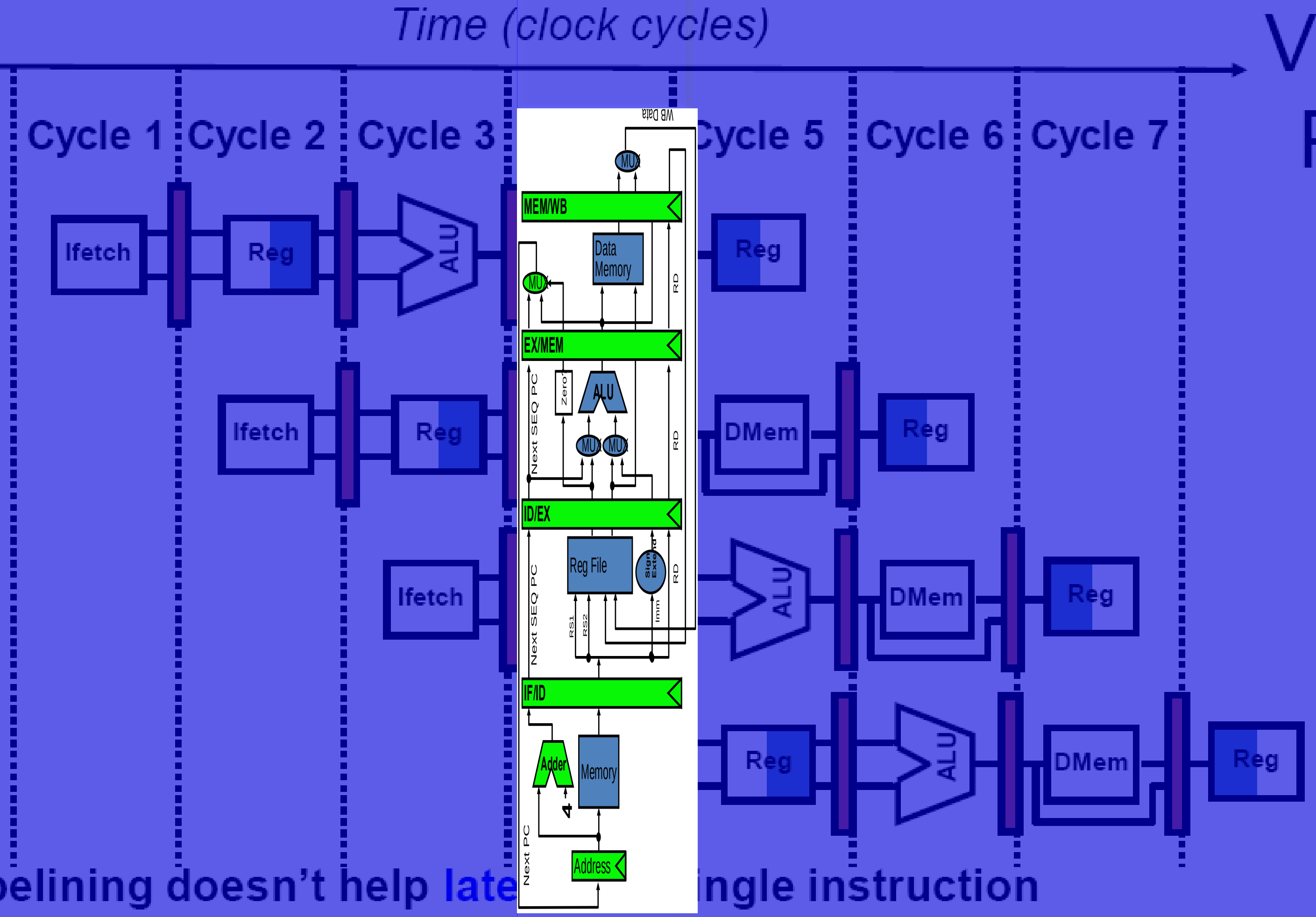
ne to "fill" pipeline and time to "drain" it reduces speedup

eedup comes from parallelism - for free – no new hardware

ny pipelines are more complex - Pentium 4 "Netburst" has 31 s

ge 133 , CA:AQA 2e

Vis
P



pelining doesn’t help late ingle instruction
- it helps **throughput** of entire workload

peline rate limited by **slowest** pipeline stage

tential speedup = **Number pipe stages**

balanced lengths of pipe stages reduces speedup

ne to “**fill**” pipeline and time to “**drain**” it reduces speedup

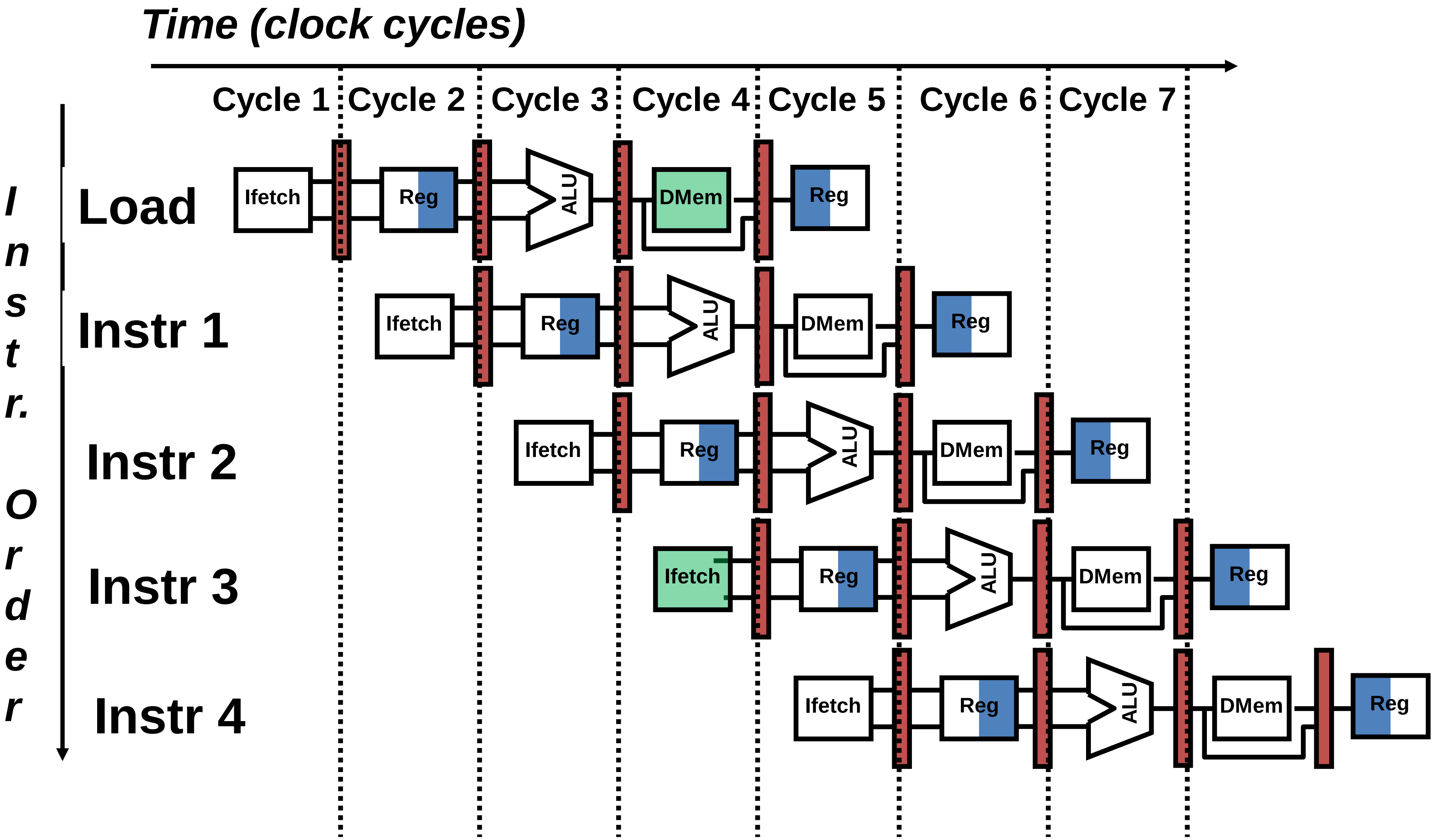
eedup comes from parallelism - for free – no new hardware

ny pipelines are more complex - Pentium 4 “Netburst” has 31 s

It’s Not That Easy

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: the hardware cannot support this combination of instructions
  - Data hazards: Instruction depends on result of a prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

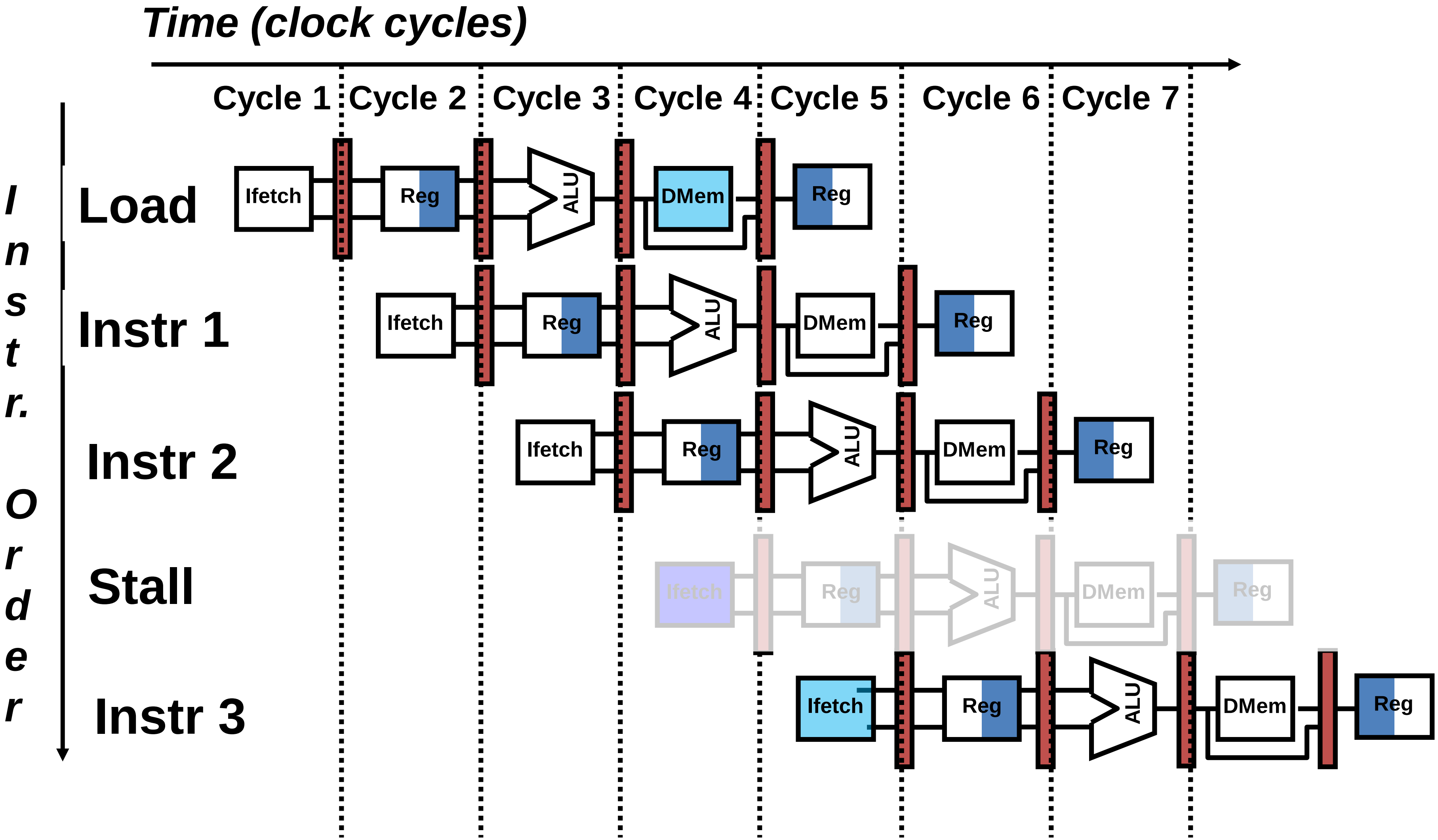
Structural Hazard: example – one RAM port



- Eg if there is only one memory for both instructions and data
- Two different stages may need access at same time
- Example: IBM/Sony/Toshiba Cell processor

Figure 3.6, Page 142 , CA:AOA 2e

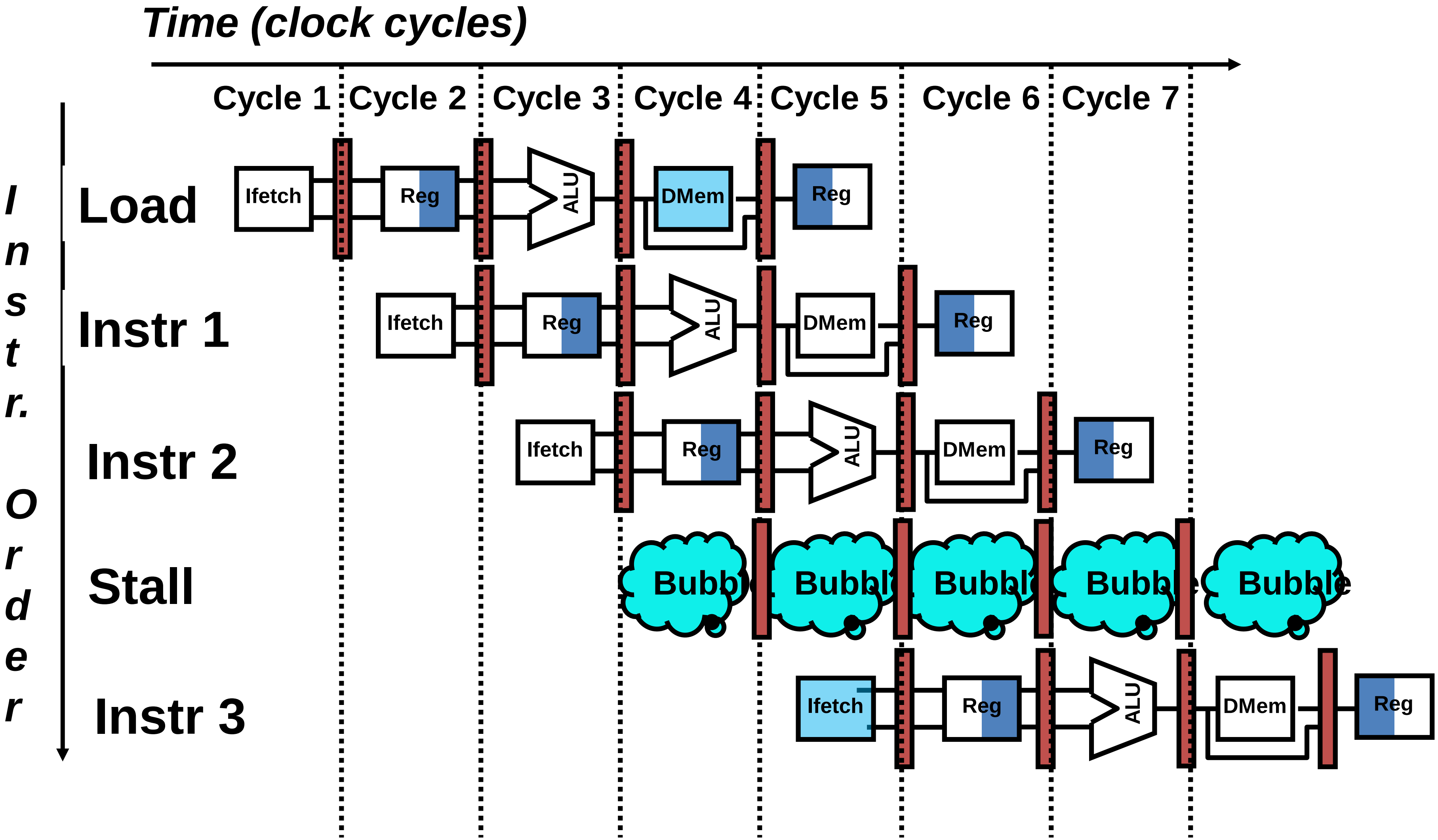
Structural Hazard: example – one RAM port



- Instr 3 cannot be loaded in cycle 4
- ID stage has nothing to do in cycle 5
- EX stage has nothing to do in cycle 6, etc. “Bubble” propagates

Figure 3.7, Page 143 , CA:AOA 2e

Structural Hazard: example – one RAM port



- Instr 3 cannot be loaded in cycle 4
- ID stage has nothing to do in cycle 5
- EX stage has nothing to do in cycle 6, etc. “Bubble” propagates

Figure 3.7, Page 143 , CA:AOA 2e

Data Hazard on R1

Time (clock cycles)

Instr. Order

add r1,r2,r3

sub r4,r1,r3

and r6,r1,r7

or r8,r1,r9

xor r10,r1,r11

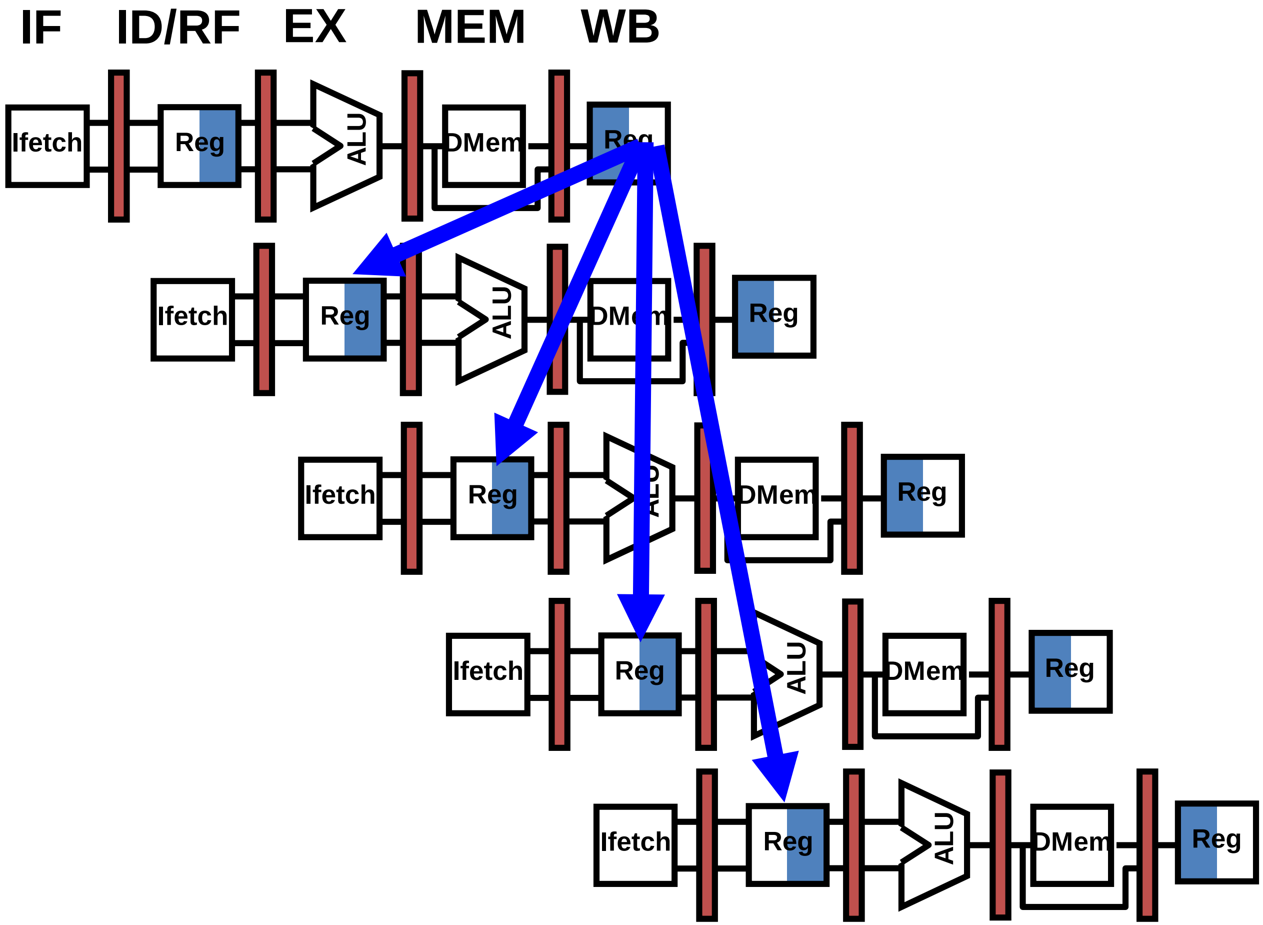
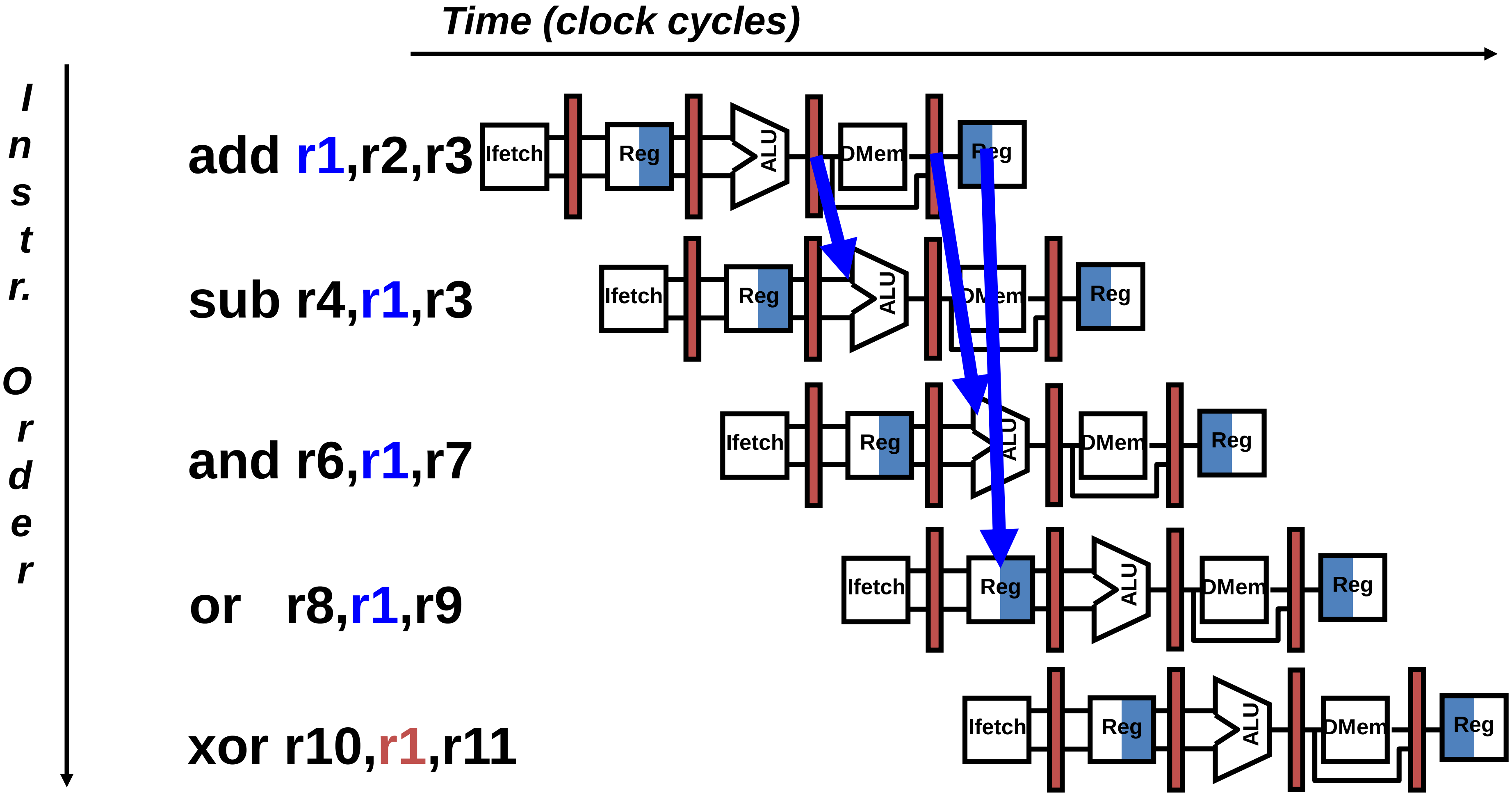


Figure 3.9, page 147 , CA:AQA 2e

Forwarding to Avoid Data Hazard

Figure 3.10, Page 149 , CA:AQA 2e



add r1,r2,r3

sub r4,r1,r3

and r6,r1,r7

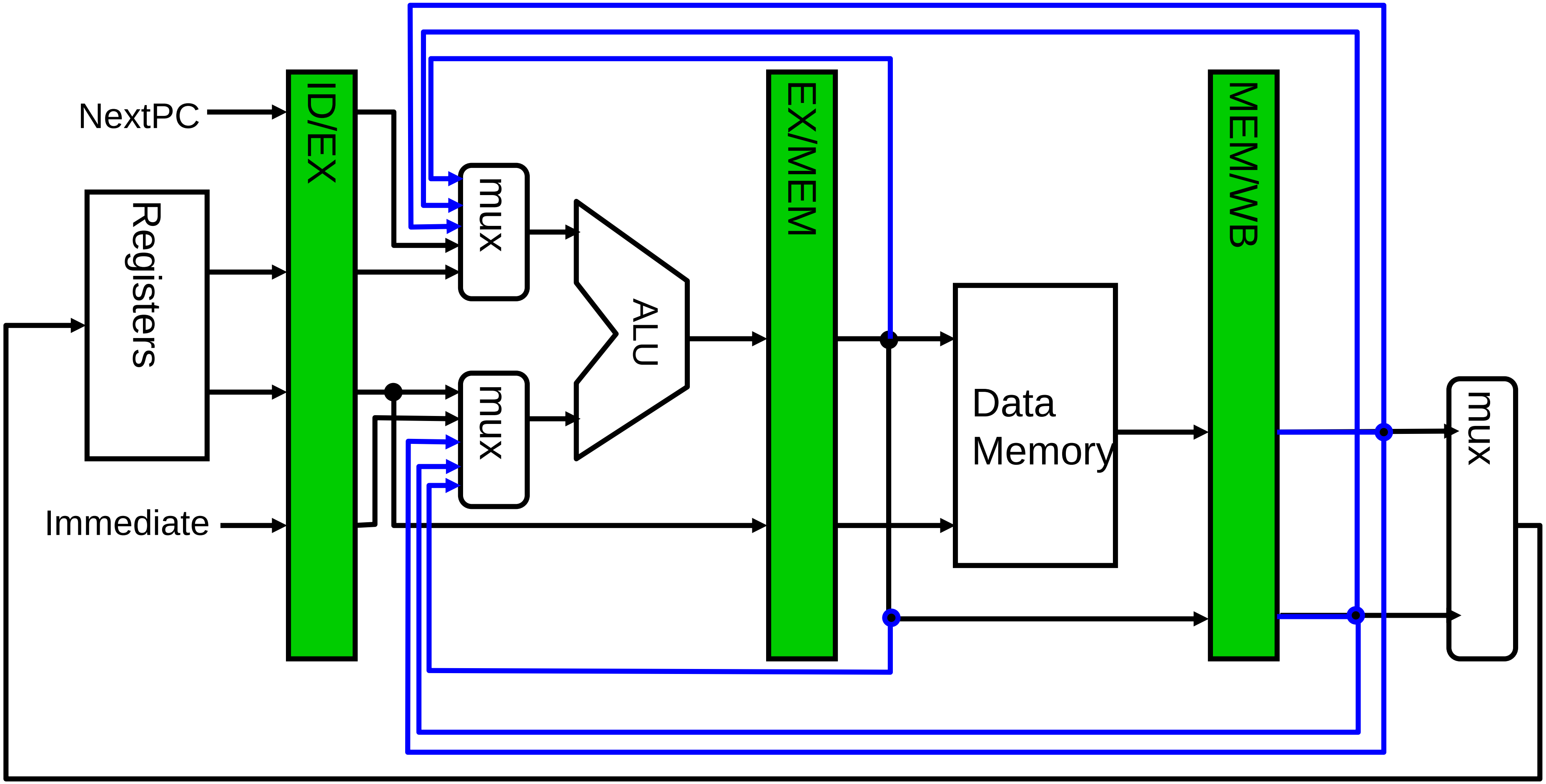
or r8,r1,r9

xor r10,r1,r11

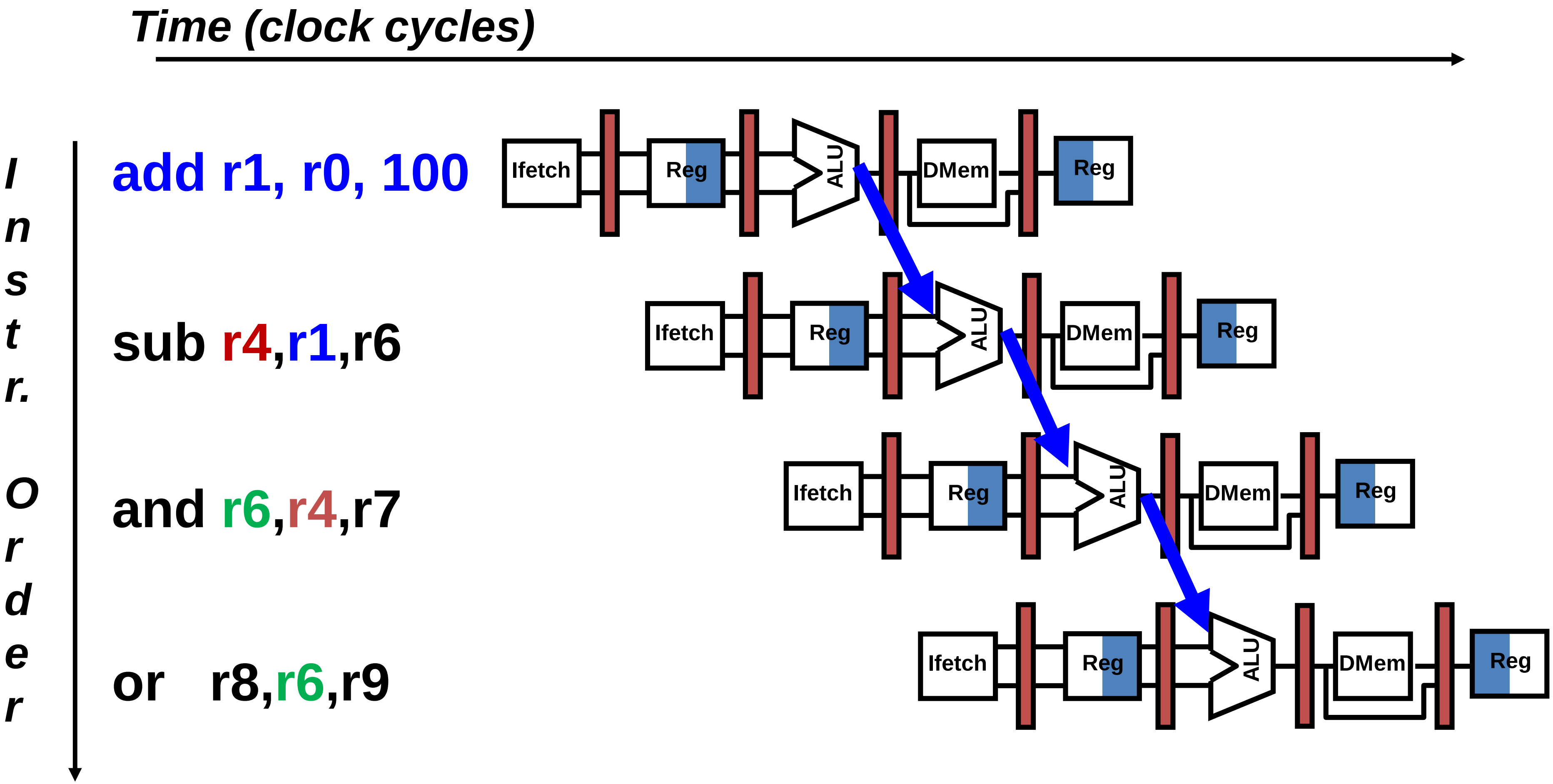
HW Change for Forwarding

Figure 3.20, Page 161, CA:AQA 2e

- Add forwarding (“bypass”) paths
- Add multiplexors to select where ALU operand should come from
- Determine mux control in ID stage
- If source register is the target of an instrn that will not WB in time



Forwarding builds the data flow graph



Time (clock cycles)

Instr. Order

add r1, r0, 100

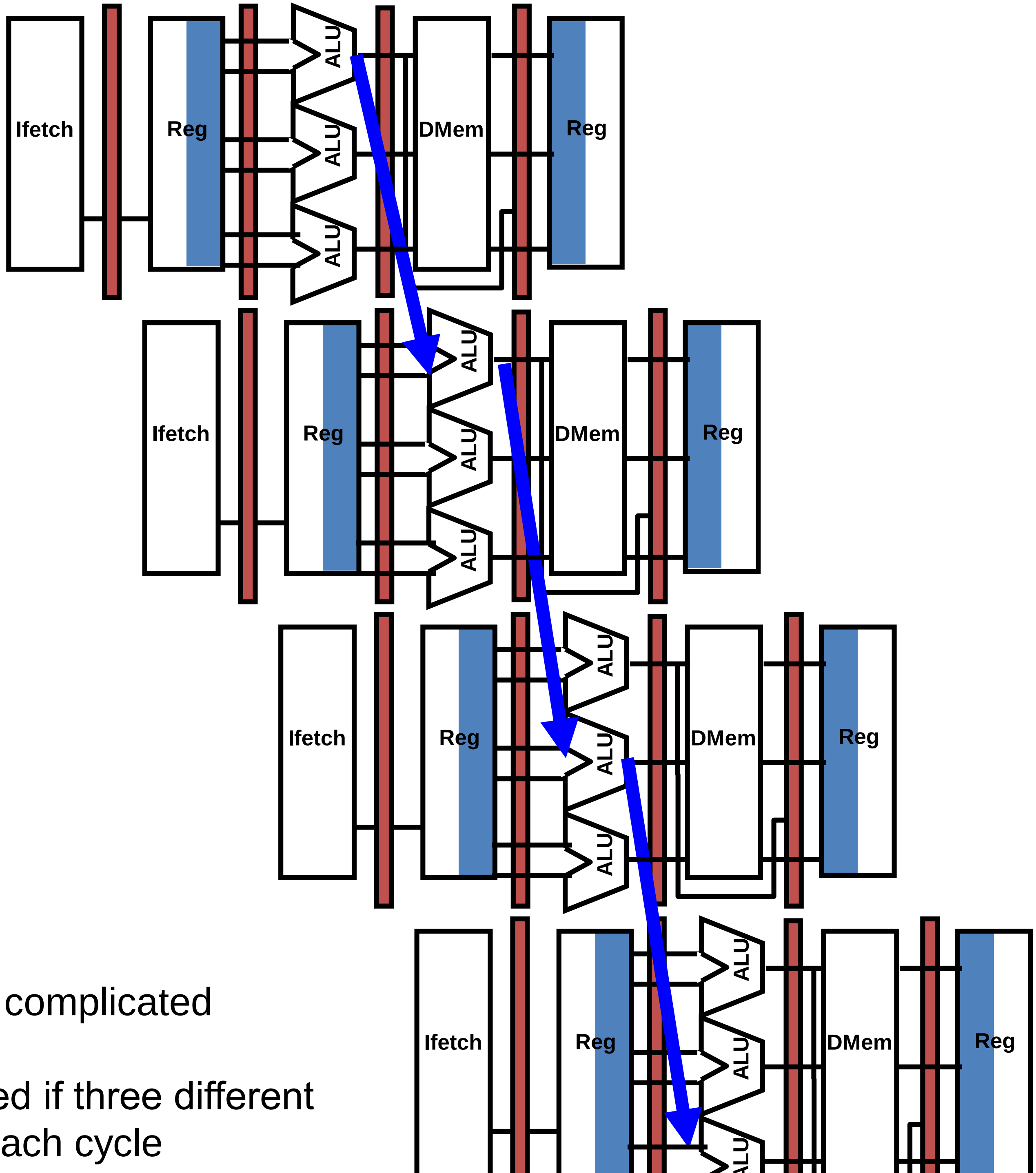
sub r4,r1,r6

and r6,r4,r7

or r8,r6,r9

- Values are passed directly from the output of the ALU to the input
- Via forwarding wires
- That are dynamically configured by the instruction decoder/control
- (This gets much more exciting when we have multiple ALUs and multiple-issue)

Imagine a machine with more ALUs



I n s t r. O r d e r

add r1, r0, 100

sub r4,r1,r6

and r6,r4,r7

or r8,r6,r9

- We would need a rather complicated forwarding network
- It’s a bit more complicated if three different instructions are issued each cycle

Data Hazard Even with Forwarding

Figure 3.12, Page 153 , CA:AQA 2e

Time (clock cycles)

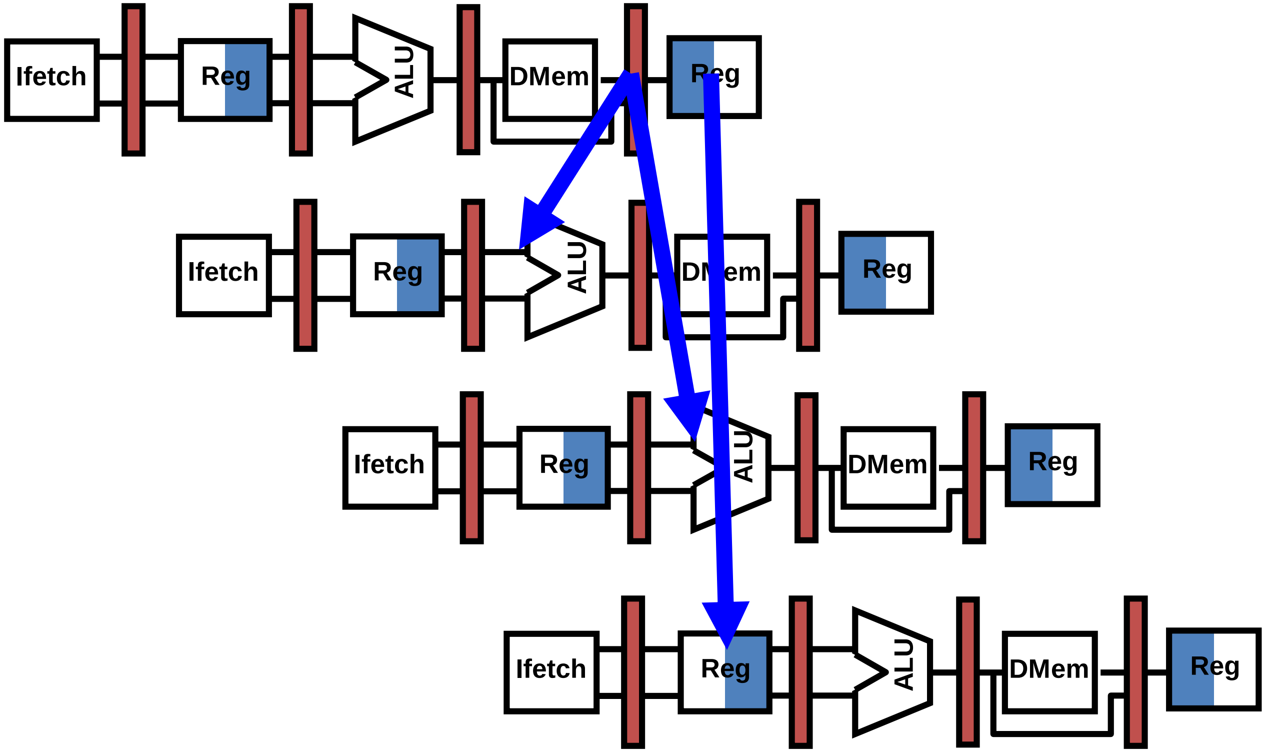
Instr. Order

lw r1, 0(r2)

sub r4,r1,r6

and r6,r1,r7

or   r8,r1,r9



Data Hazard Even with Forwarding

Figure 3.12, Page 153 , CA:AQA 2e

Time (clock cycles)

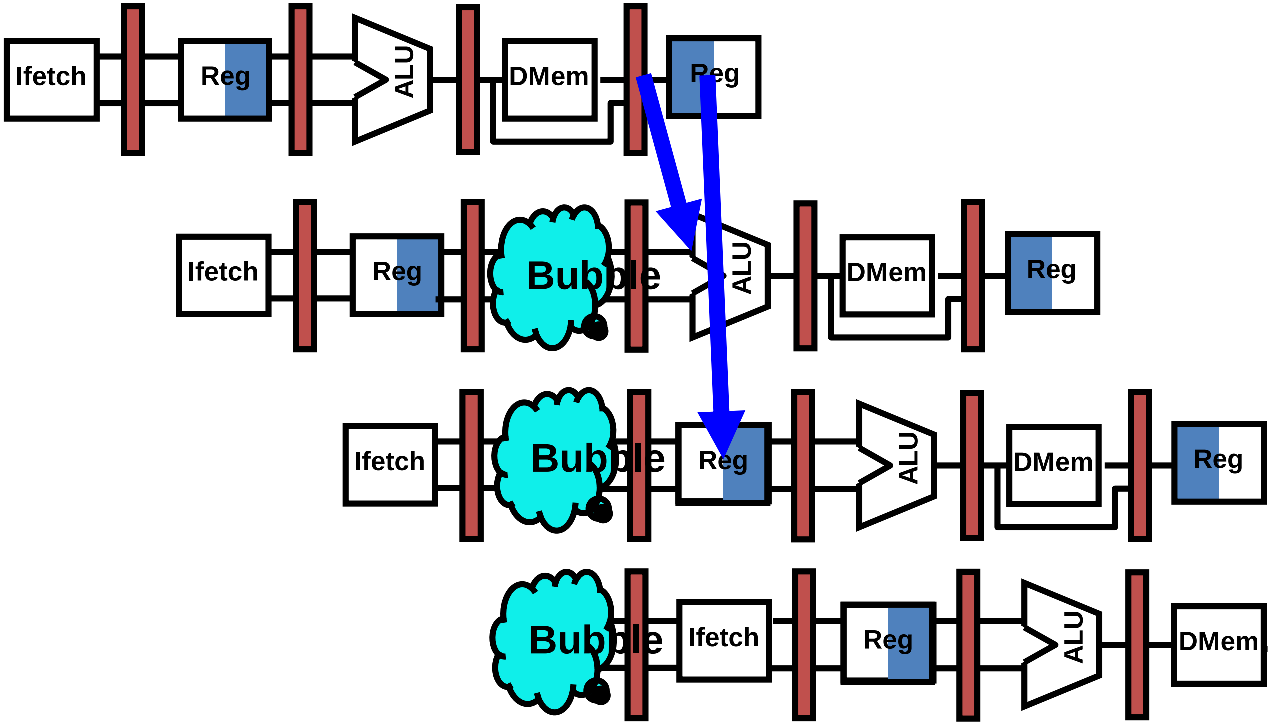
Instr. Order

lw r1, 0(r2)

sub r4,r1,r6

and r6,r1,r7

or r8,r1,r9



EX stage waits in cycle 4 for operand
Following instruction (“and”) waits in ID stage
Missed instruction issue opportunity…

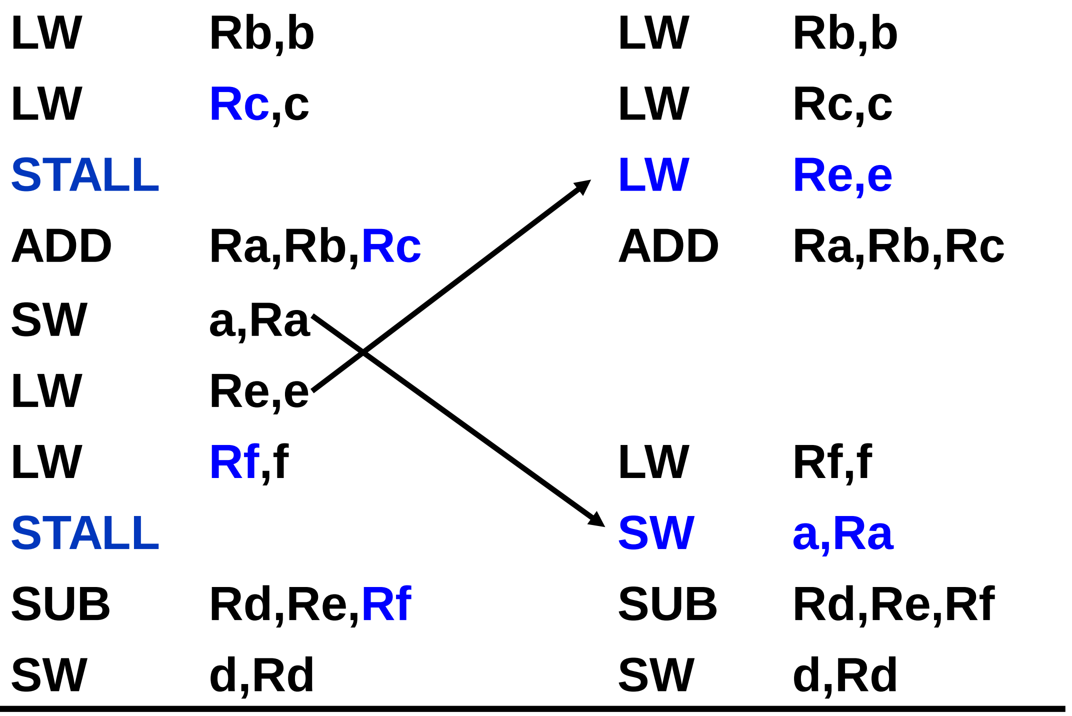
Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.



Slow code:		Fast code:	
LW	**Rb,b**	**LW**	**Rb,b**
LW	**Rc,c**	**LW**	**Rc,c**
STALL		**LW**	**Re,e**
ADD	**Ra,Rb,Rc**	**ADD**	**Ra,Rb,Rc**
SW	**a,Ra**		
LW	**Re,e**		
LW	**Rf,f**	**LW**	**Rf,f**
STALL		**SW**	**a,Ra**
SUB	**Rd,Re,Rf**	**SUB**	**Rd,Re,Rf**
SW	**d,Rd**	**SW**	**d,Rd**
10 cycles (2 stalls)		8 cycles (0 stalls)	

Show the stalls explicitly

Control Hazard on Branches

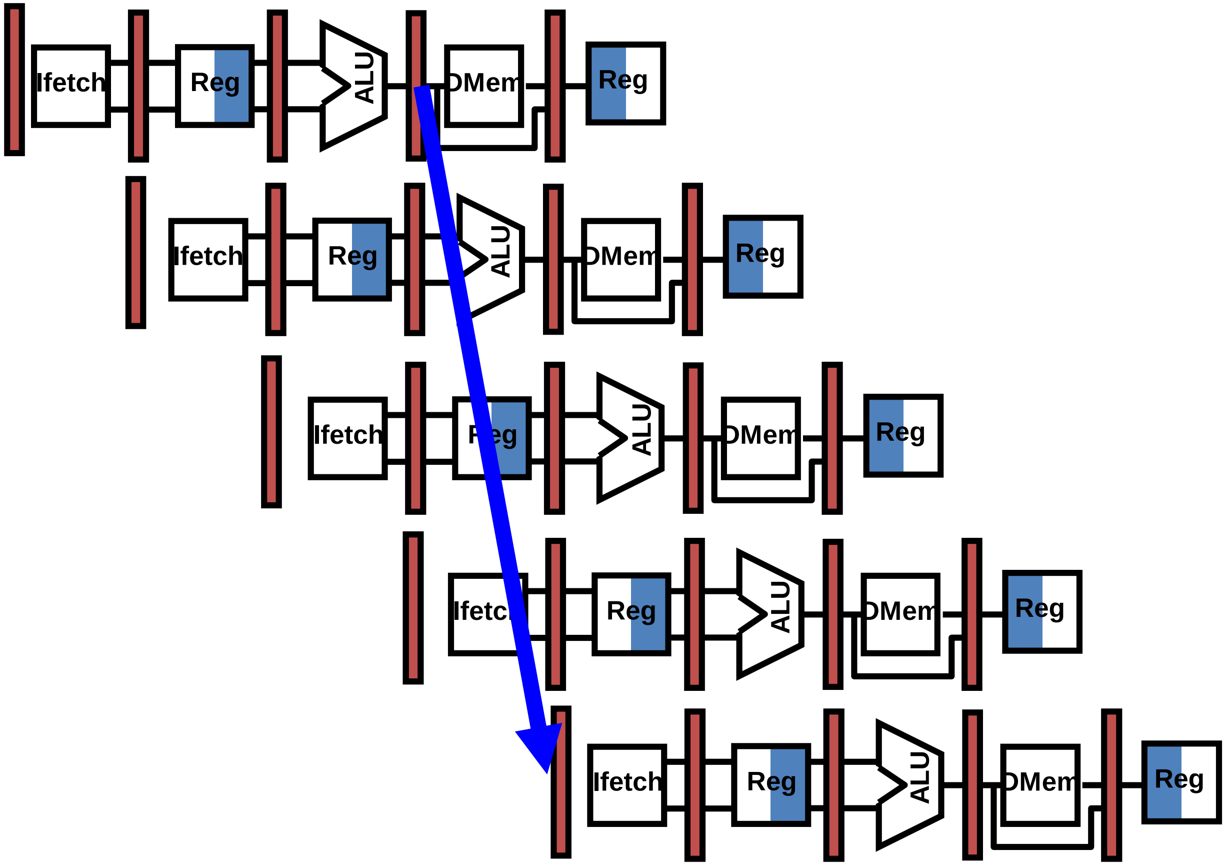
10: beq r1,r3,36

14: and r2,r3,r5

18: or r6,r1,r7

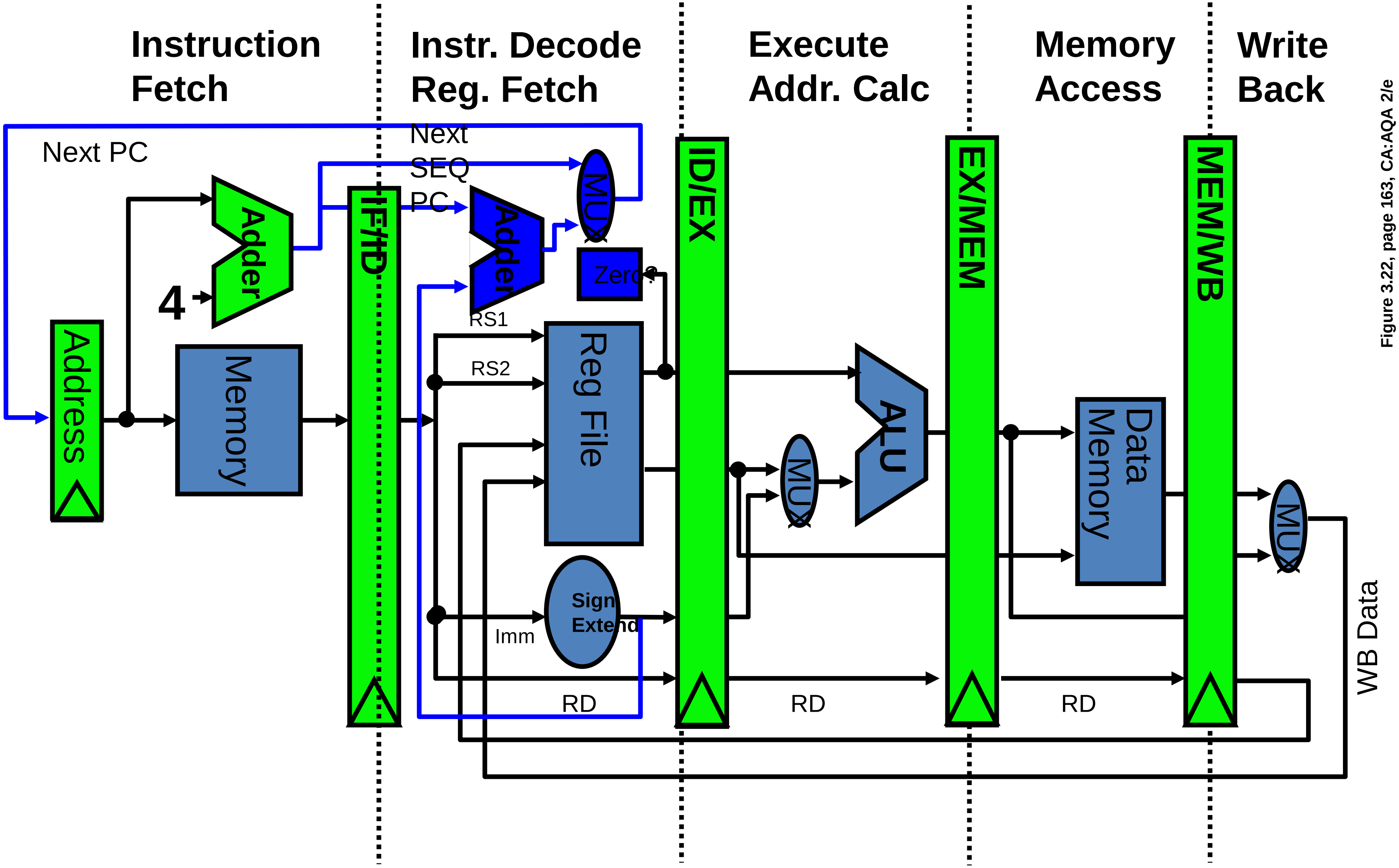
22: add r8,r1,r9

36: xor r10,r1,r11



If we're not smart we risk a three-cycle stall

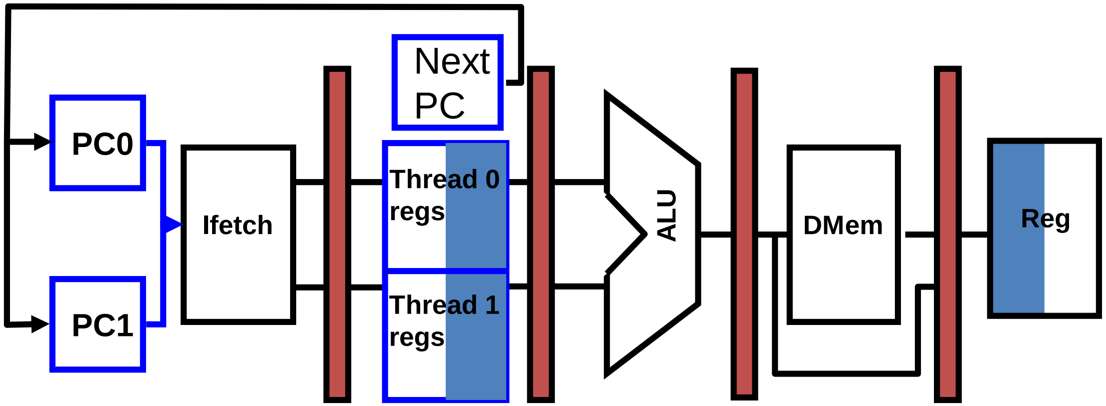
Pipelined MIPS Datapath with early branch determination



- Add extra hardware to the decode stage, to determine branch direction and target earlier
- We still have a one-cycle delay – we just have to fetch and start executing the next instruction
- If the branch is actually taken, block the MEM and WB stages and fetch the right instruction

Eliminating hazards with simultaneous multi-threading

- If we had no stalls we could finish one instruction every cycle
- If we had no hazards we could do without forwarding – and decode/control would be simpler too



Example: PowerPC processing element (PPE) in the Cell Broadband Engine (Sony PlayStation 3)

- **IF maintains two Program Counters**
- **Even cycle – fetch from PC0**
- **Odd cycle – fetch from PC1**
- **Thread 0 reads and writes thread-0 registers**
- **No register-to-register hazards between adjacent pipeline stages**

(cf "C-Slowing"......)

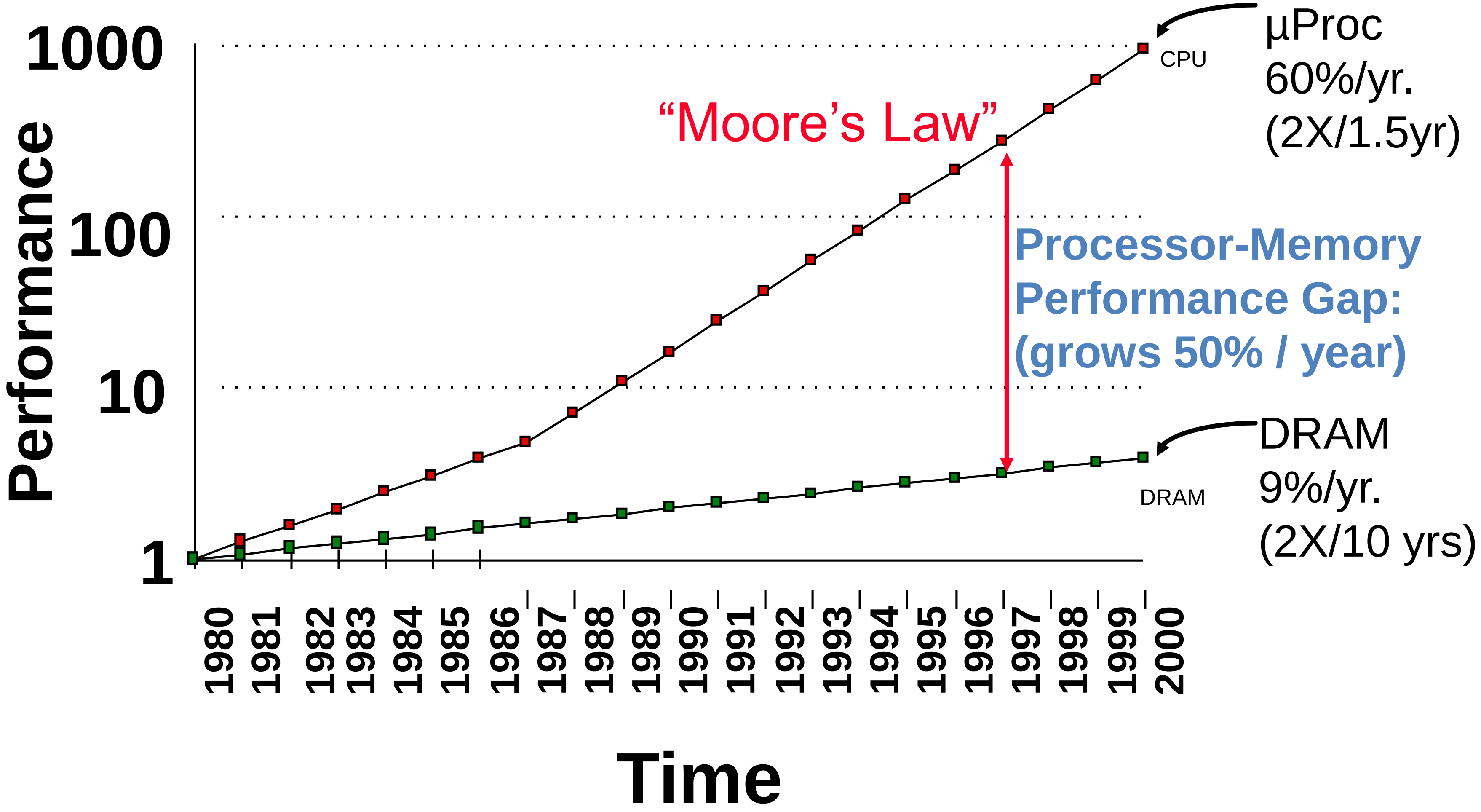
So – how fast can this design go?

- A simple 5-stage pipeline can run at 5-9GHz
- Limited by critical path through slowest pipeline stage logic
- Tradeoff: do more per cycle? Or increase clock rate?
  - Or do more per cycle, in parallel…
- At 3GHz, clock period is 330 picoseconds.
  - The time light takes to go about four inches
  - About 10 gate delays
    - for example, the Cell BE is designed for 11 FO4 ("fan-out=4") gates per cycle: www.fe.infn.it/~belletti/articles/ISSCC2005-cell.pdf
    - Pipeline latches etc account for 3-5 FO4 delays leaving only 5-8 for actual **work**

- **How can we build a RAM that can implement our MEM stage in 5-8 FO4 delays?**

Life used to be so easy

Processor-DRAM Memory Gap (latency)



In 1980 a large RAM's access time was close to the CPU cycle time. 1980s machines had little or no need for cache. Life is no longer quite so simple.

Levels of the Memory Hierarchy

Capacity
Access Time
Cost

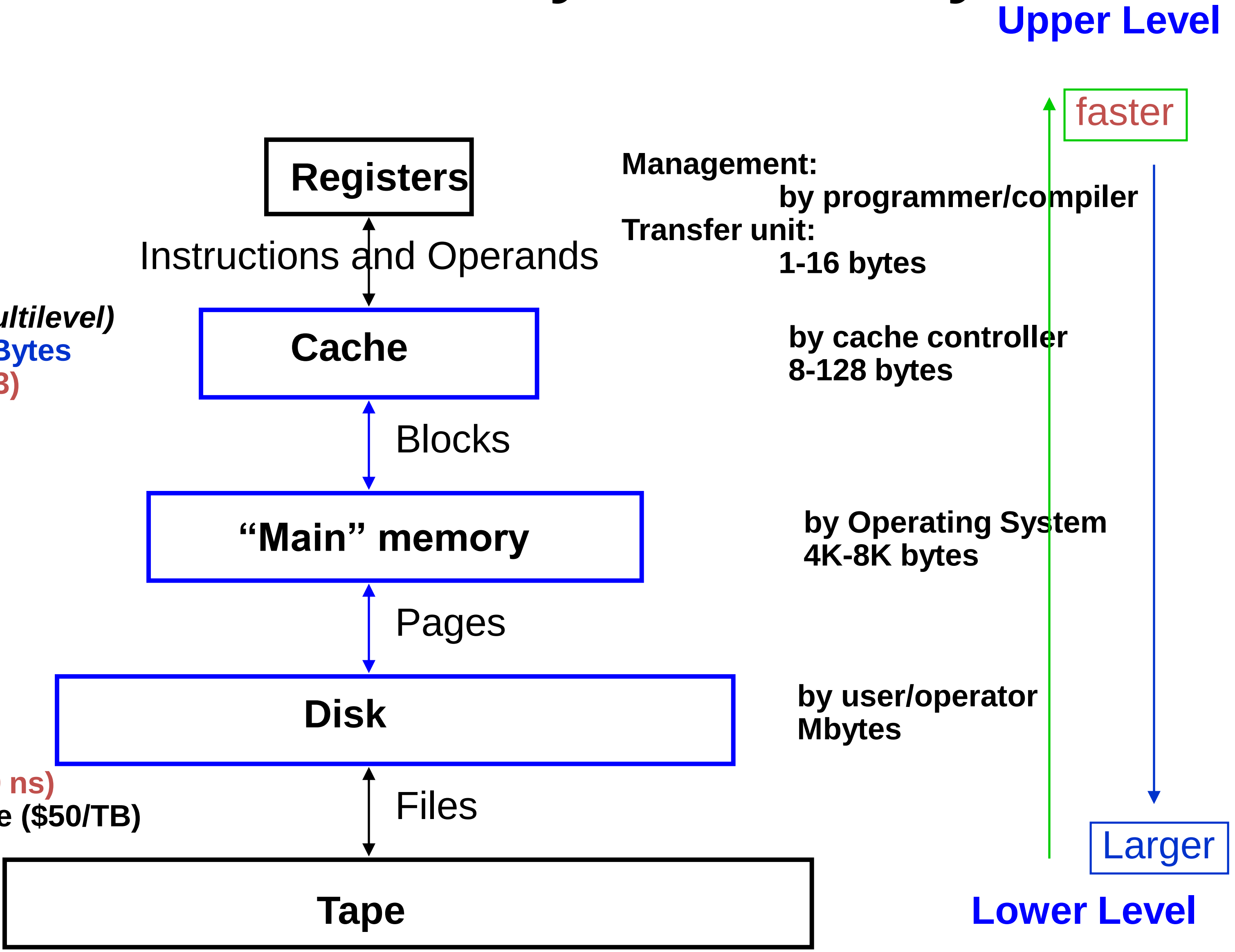
CPU Registers
100s Bytes
<1ns

Cache (perhaps multilevel)
10s-1000s K Bytes
1-10 ns (L1-L3)
~$10/ MByte

Main Memory
G Bytes
100ns- 300ns
$0.01/ MByte

Disk
100s G Bytes,
10 ms
(10,000,000 ns)
$0.00005 Mbyte ($50/TB)

Tape
infinite
sec-min
$0.00005/ MByte



Upper Level

faster

Registers

Instructions and Operands

Cache

Blocks

"Main" memory

Pages

Disk

Files

Tape

Larger

Lower Level

Management:
by programmer/compiler
Transfer unit:
1-16 bytes

by cache controller
8-128 bytes

by Operating System
4K-8K bytes

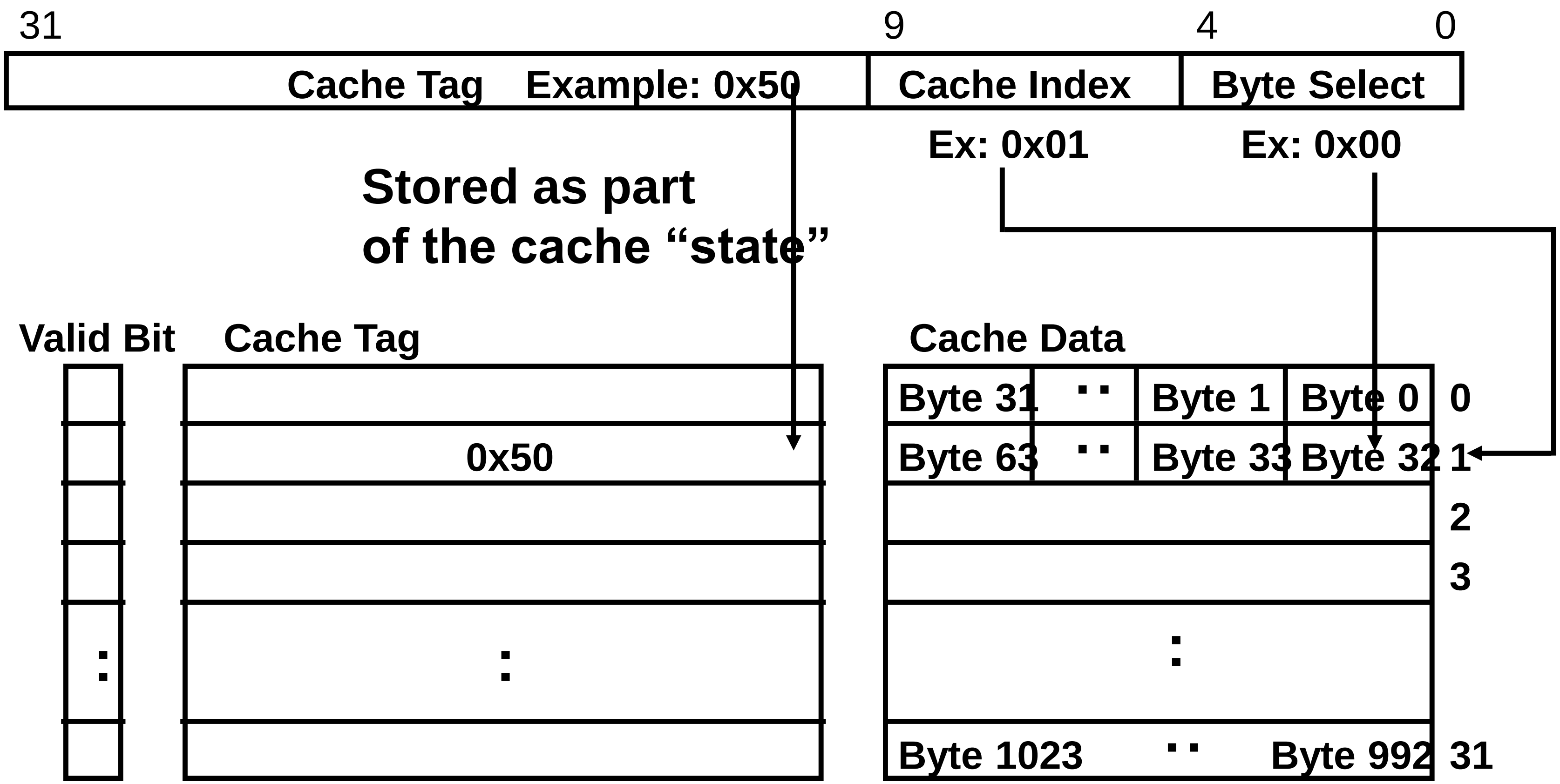
by user/operator
Mbytes

~ Exponential increase in access latency, block size, capacity

- The Principle of Locality:
  - Programs access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon
    (e.g., straightline code, array access)
- Most modern architectures are heavily reliant (totally reliant?) on locality for speed

1 KB “Direct Mapped” Cache, 32B blocks

- For a 2^N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2^M)



31 …	9 …	4 … 0
Cache Tag Example: 0x50	Cache Index	Byte Select
	Ex: 0x01	Ex: 0x00

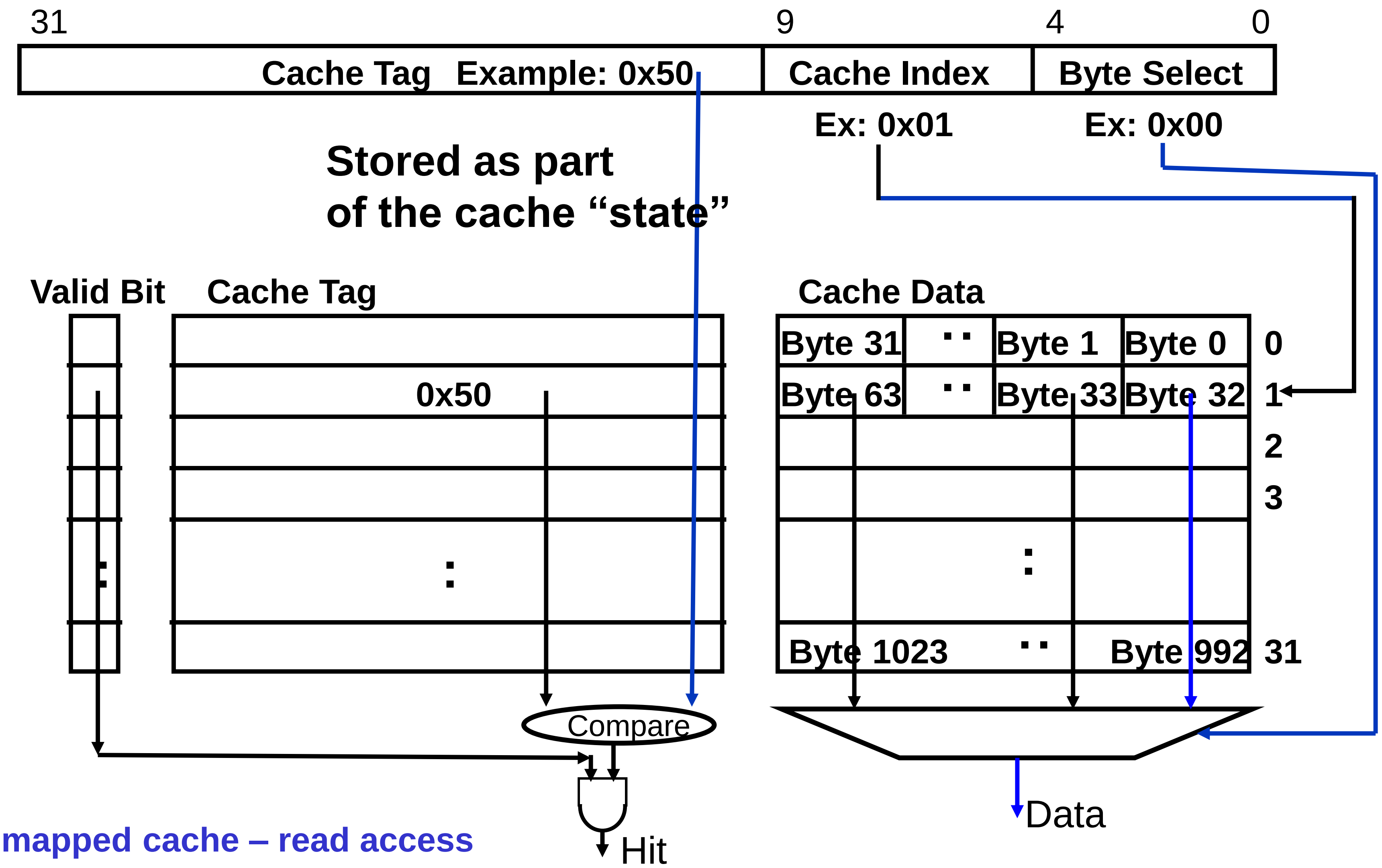
Stored as part
of the cache “state”

Valid Bit	Cache Tag	Cache Data				
		Byte 31	:	Byte 1	Byte 0	0
	0x50	Byte 63	:	Byte 33	Byte 32	1
						2
						3
:	:	:				
		Byte 1023	:		Byte 992	31

Direct-mapped cache - storage

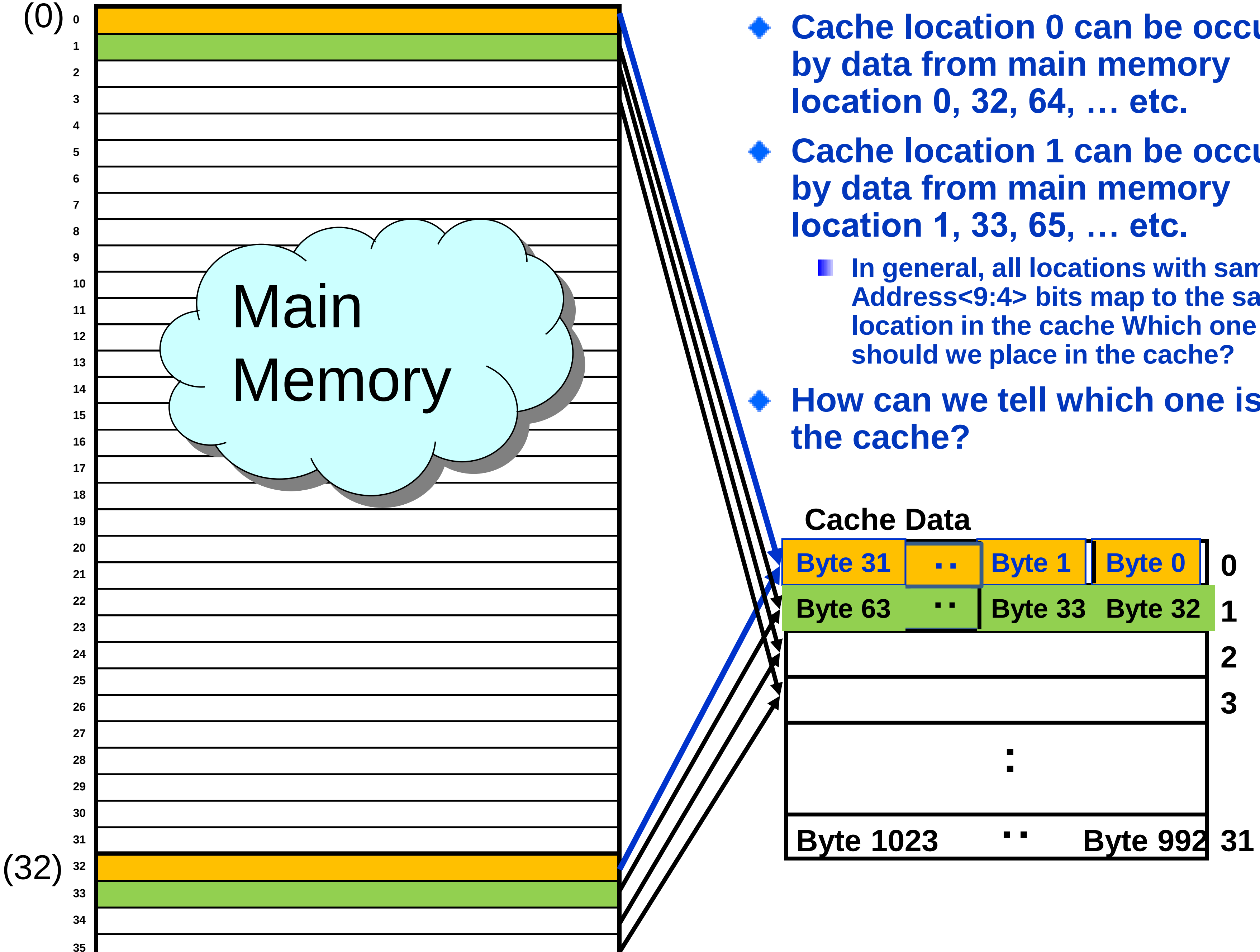
1 KB "Direct Mapped" Cache, 32B blocks

- For a 2^N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2^M)



Direct-mapped cache – read access

1 KB Direct Mapped Cache, 32B blocks



- **Cache location 0 can be occupied by data from main memory location 0, 32, 64, … etc.**
- **Cache location 1 can be occupied by data from main memory location 1, 33, 65, … etc.**
  - **In general, all locations with same Address<9:4> bits map to the same location in the cache Which one should we place in the cache?**
- **How can we tell which one is in the cache?**

Associativity conflicts in a direct-mapped cache

- **Consider a loop that repeatedly reads part of two different arrays:**

```
int A[256];
int B[256];
int r = 0;
for (int i=0; i<10; ++i) {
  for (int j=0; j<64; ++j) {
    r += A[j] + B[j];
  }
}
```

For the accesses to A and B to be mostly cache hits, we need a cache big enough to hold 2x64 ints, ie 512B

Consider the 1KB direct-mapped cache on the previous slide - what might go wrong?

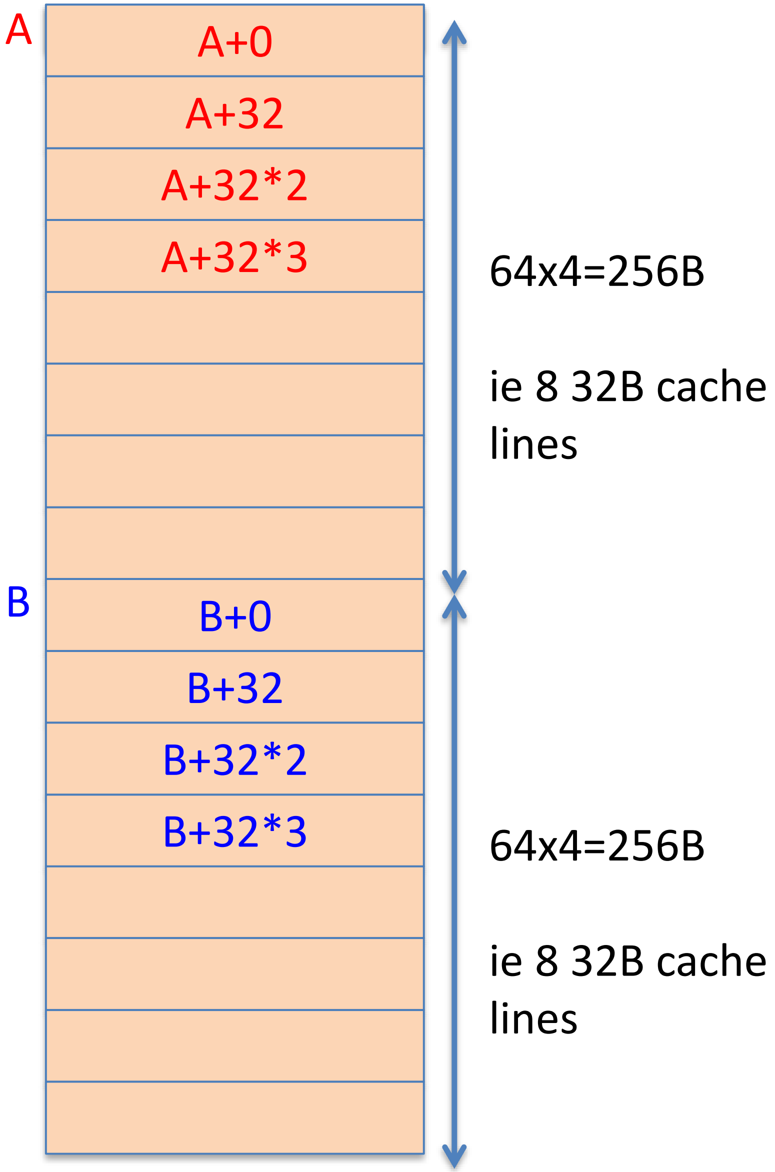
Associativity conflicts in a direct-mapped cache

- **Consider a loop that repeatedly reads part of two different arrays:**

```
int A[256];
int B[256];
int r = 0;
for (int i=0; i<10; ++i) {
  for (int j=0; j<64; ++j) {
    r += A[j] + B[j];
  }
}
```

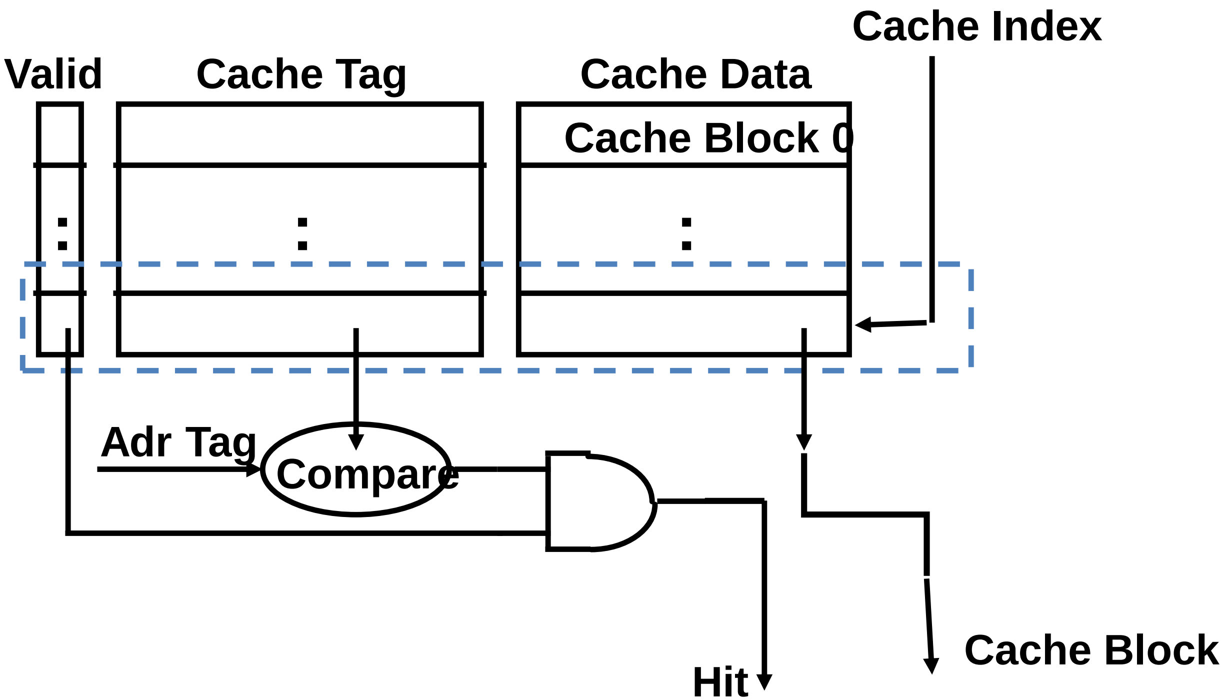
For the accesses to A and B to be mostly cache hits, we need a cache big enough to hold 2x64 ints, ie 512B

Consider the 1KB direct-mapped cache on the previous slide - what might go wrong?



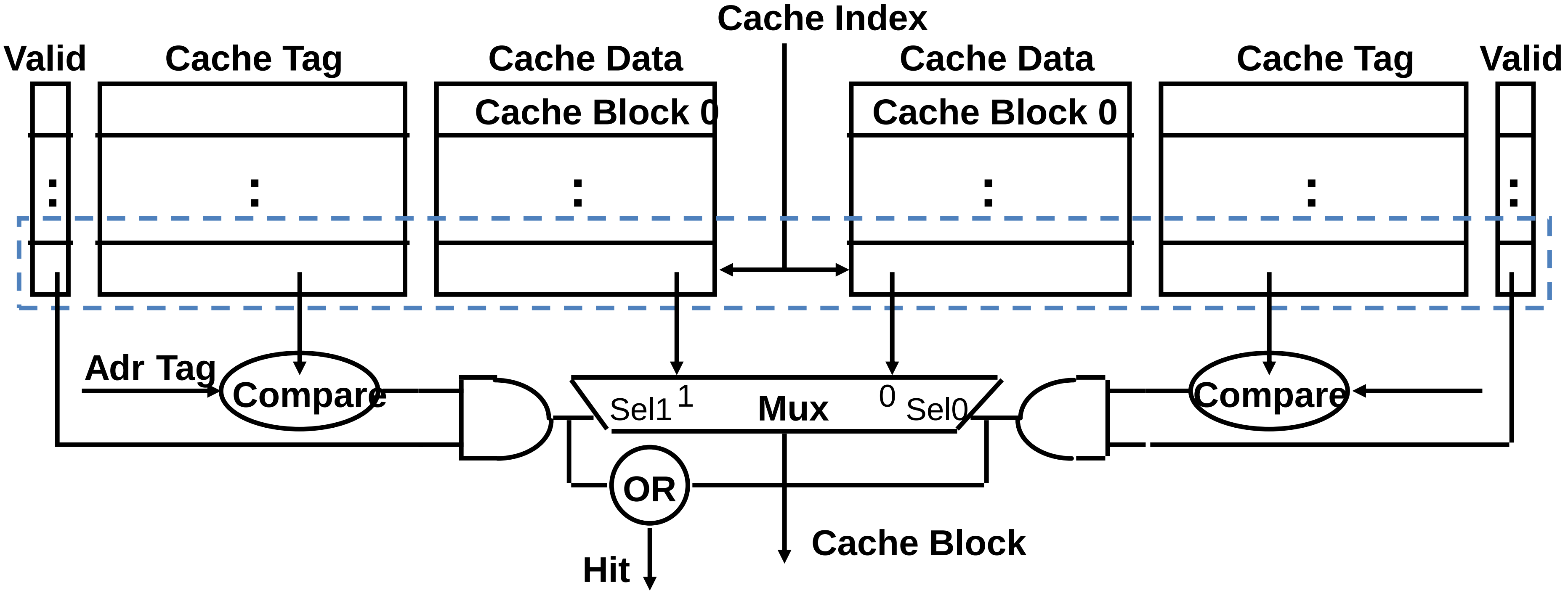
Direct-mapped Cache - structure

- Capacity: C bytes (eg 1KB)
- Blocksize: B bytes (eg 32)
- Byte select bits: 0..log(B)-1 (eg 0..4)
- Number of blocks: C/B (eg 32)
- Address size: A (eg 32 bits)
- Cache index size: I=log(C/B) (eg log(32)=5)
- Tag size: A-I-log(B) (eg 32-5-5=22)



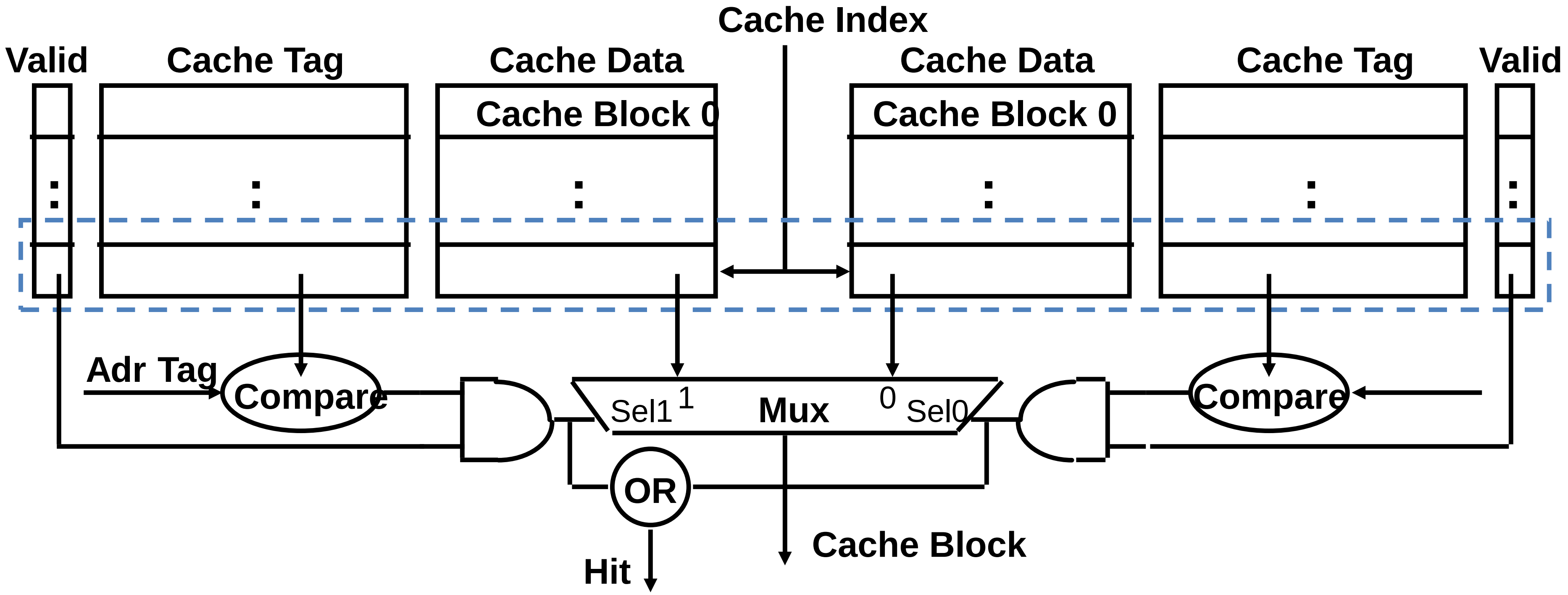
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operated in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



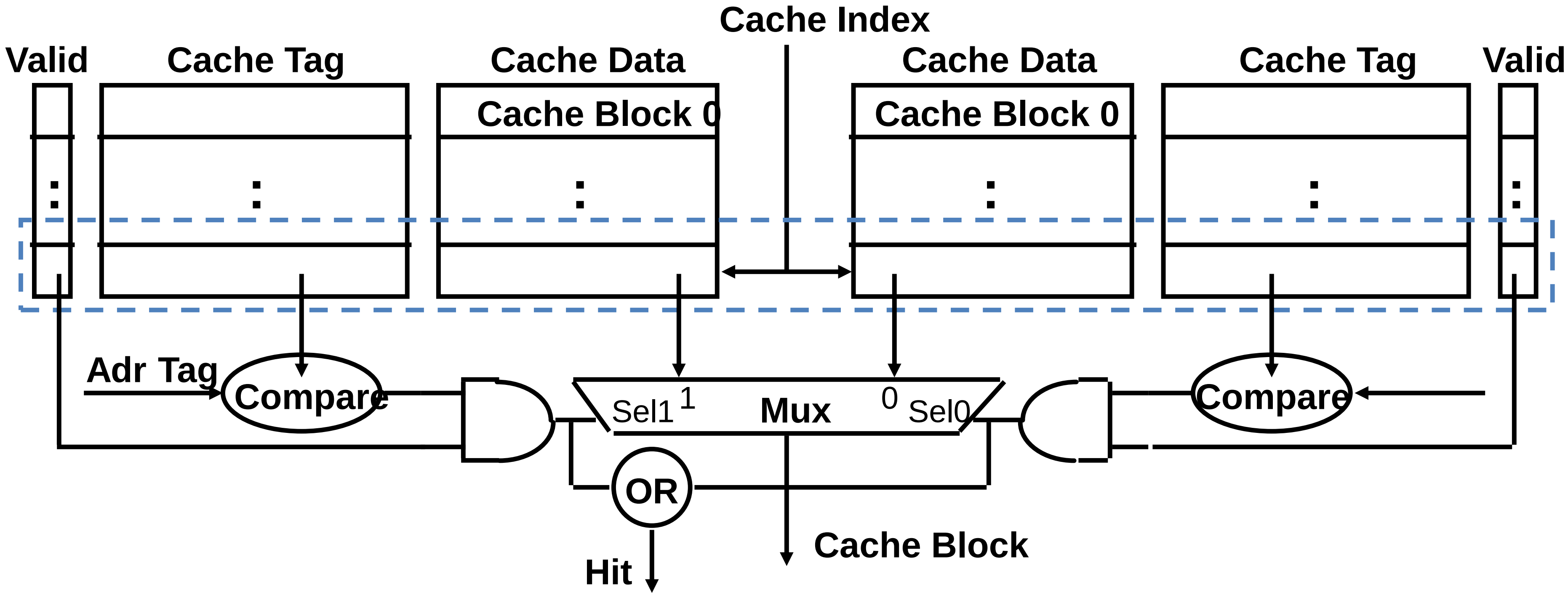
Disadvantage of Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



Example: Intel Pentium 4 Level-1 cache (pre-Prescott)

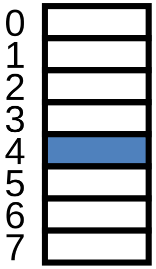
- **Capacity: 8K bytes** (total amount of data cache can store)
- **Block: 64 bytes** (so there are 8K/64=128 blocks in the cache)
- **Ways: 4** (addresses with same index bits can be placed in one of 4 ways)
- **Sets: 32** (=128/4, that is each RAM array holds 32 blocks)
- **Index: 5 bits** (since 2^5=32 and we need index to select one of the 32 ways)
- **Tag: 21 bits** (=32 minus 5 for index, minus 6 to address byte within block)
- **Access time: 2 cycles,** (.6ns at 3GHz; pipelined, dual-ported [load+store])



4 Questions for Memory Hierarchy

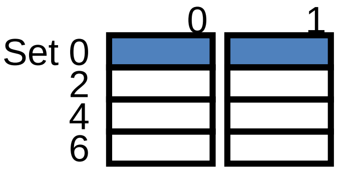
- Q1: Where can a block be placed in the upper level?
  - *Block placement*
- Q2: How is a block found if it is in the upper level?
  - *Block identification*
- Q3: Which block should be replaced on a miss?
  - *Block replacement*
- Q4: What happens on a write?
  - *Write strategy*

Q1: Where can a block be placed in the upper level?



In a direct-mapped cache, block 12 can only be placed in one cache location, determined by its low-order address bits –

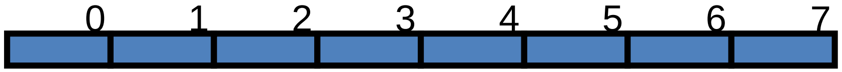
(12 mod 8) = 4



In a two-way set-associative cache, the set is determined by its low-order address bits –

(12 mod 4) = 0

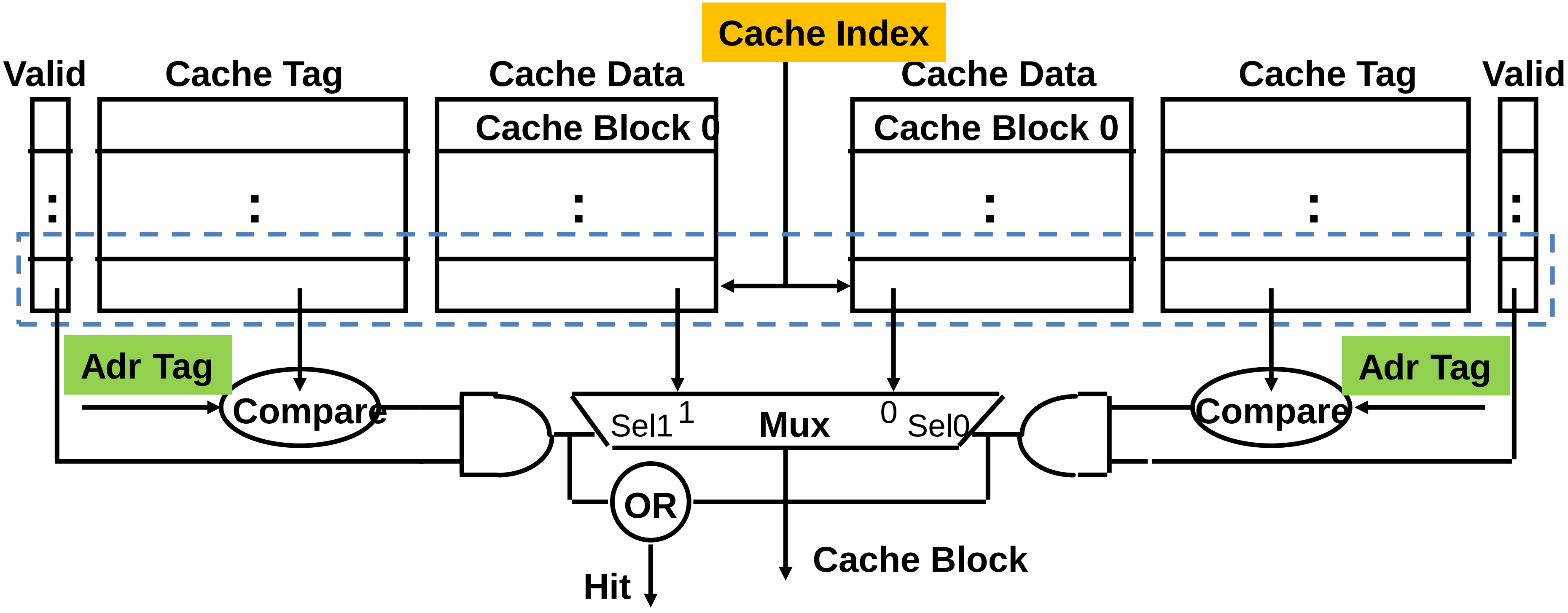
Block 12 can be placed in either of the two cache locations in set 0



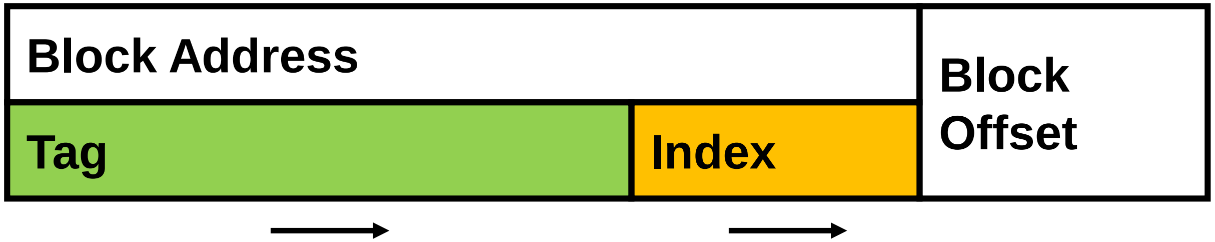
In a fully-associative cache, block 12 can be placed in any location in the cache

- **More associativity:**
  - **More comparators – larger, more energy**
  - **Better hit rate (diminishing returns)**
  - **Reduced storage layout sensitivity – more predictable**

Q2: How is a block found if it is in the upper level?



- Tag on each block
  - No need to check index or block offset



- Increasing associativity shrinks index, expands tag

Q3: Which block should be replaced on a miss?

- With Direct Mapped there is no choice
- With Set Associative or Fully Associative we want to choose
  - Ideal: least-soon re-used
  - LRU (Least Recently Used) is a popular approximation
  - Random is remarkably good in large caches

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Benchmark studies show that LRU beats random only with small caches

LRU can be pathologically bad.......

Q4: What happens on a write?

- *Write through*—The information is written to both the block in the cache and to the block in the lower-level memory

- *Write back*—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - **is block clean or dirty?**

- Pros and Cons of each?
  - **WT: read misses cannot result in writes**
  - **WB: no repeated writes to same location**

- WT always combined with write buffers so that don't wait for lower level memory

Caches are a *big* topic

- Cache coherency
  - If your data can be in more than one cache, how do you keep the copies consistent?
- Victim caches
  - Stash recently-evicted blocks in a small fully-associative cache (a “competitive strategy”)
- Prefetching
  - Use a predictor to guess which block to fetch next – *before* the processor requests it
- And much much more........

What’s at the bottom of the memory hierarchy?

- StorageTek STK 9310 (“Powderhorn”)
  - 2,000, 3,000, 4,000, 5,000, or 6,000 cartridge slots per library storage module (LSM)
  - Up to 24 LSMs per library (144,000 cartridges)
  - 120 TB (1 LSM) to 28,800 TB capacity (24 LSM)
  - Each cartridge holds 300GB, readable up to 40 MB/sec
- Up to 28.8 petabytes
- Ave 4s to load tape

- 2017 product: Oracle SL8500
- Up to 1.2 Exabyte per unit
- Combine up to 32 units into single robot tape drive system
- http://www.oracle.com/us/products/servers-storage/storage/tape-storage/034341.pdf

https://www.itnews.com.au/gallery/inside-suns-multi-storey-colorado-data-centre-135385/page1

Can we live without cache?

http://www.karo.com

- Interesting exception: Cray/Tera MTA, first delivered June 1999:
  - www.cray.com/products/systems/mta/

- Each CPU switches every cycle between 128 threads

- Each thread can have up to 8 outstanding memory accesses

- 3D toroidal mesh interconnect

- Memory accesses hashed to spread load across banks

- MTA-1 fabricated using Gallium Arsenide, not silicon
- “nearly un-manufacturable” (wikipedia)

- Third-generation Cray XMT:
  - http://www.cray.com/Products/XMT.aspx
  - YarcData's uRiKA (http://www.yarcdata.com/products.html)