EECS 322 Computer Architecture

Improving Memory Access: the Cache

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please viewshow

Review: Models

Single-cycle model (non-overlapping)

- The instruction latency executes in a single cycle
- Every instruction and clock-cycle must be stretched to the slowest instruction (p.438)

Multi-cycle model (non-overlapping)

- The instruction latency executes in multiple-cycles
- The clock-cycle must be stretched to the slowest step
- Ability to share functional units within the execution of a single instruction

Pipeline model (overlapping, p. 522)

- The instruction latency executes in multiple-cycles
- The clock-cycle must be stretched to the slowest step
- The throughput is mainly one clock-cycle/instruction
- Gains efficiency by overlapping the execution of multiple instructions, increasing hardware utilization. (p. 377)

Review: Pipeline Hazards

Pipeline hazards

- **Solution #1** always works (for non-realtime) applications: **stall.**

Structural Hazards (i.e. fetching same memory bank)

- **Solution #2: partition architecture**

Control Hazards (i.e. branching)

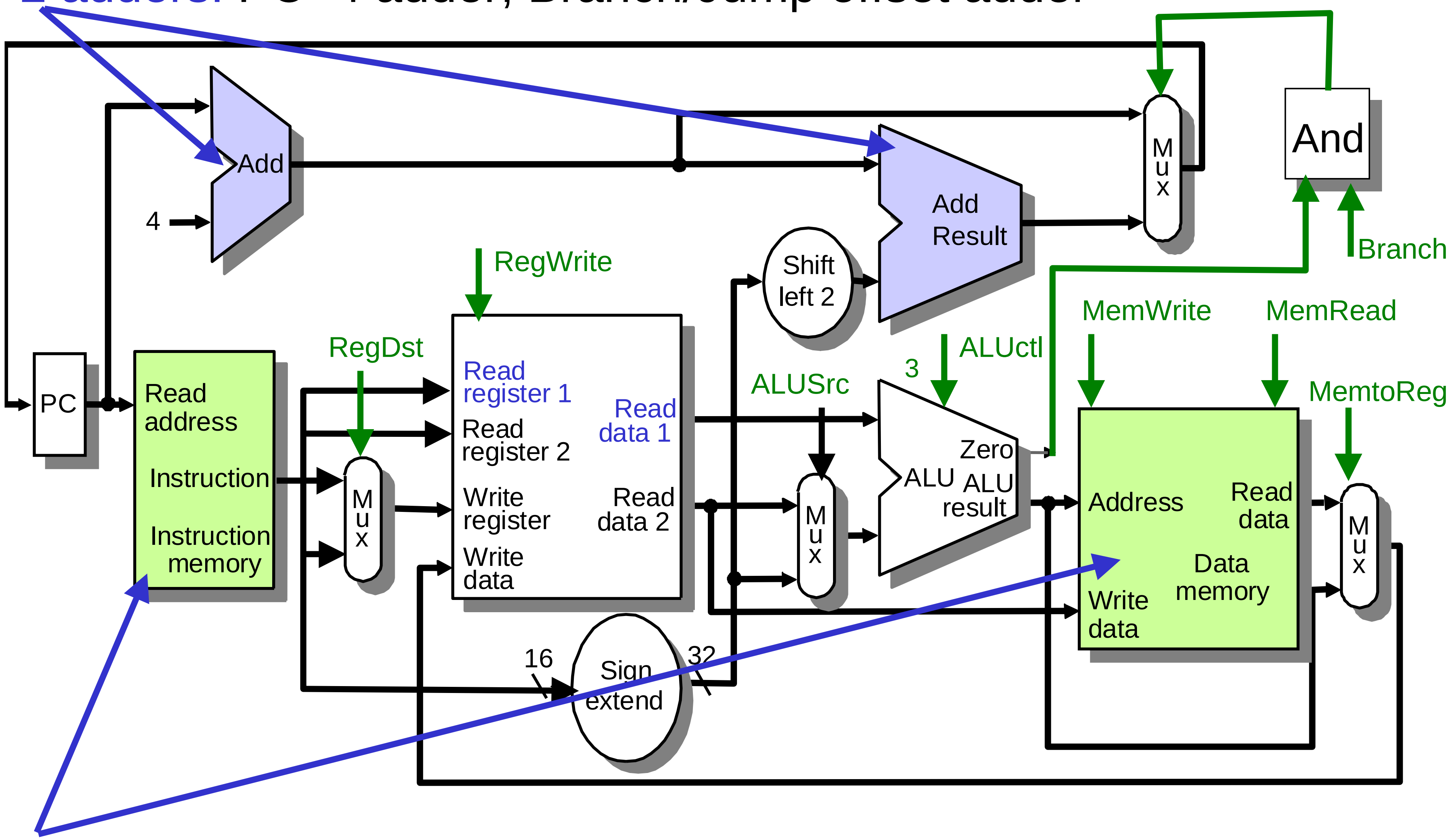
- **Solution #1: stall! but decreases throughput**
- **Solution #2: guess and back-track**
- **Solution #3: delayed decision: delay branch & fill slot**

Data Hazards (i.e. register dependencies)

- **Worst case situation**
- **Solution #2: re-order instructions**
- **Solution #3: forwarding or bypassing: delayed load**

Review: Single-Cycle Datapath

2 adders: PC+4 adder, Branch/Jump offset adder

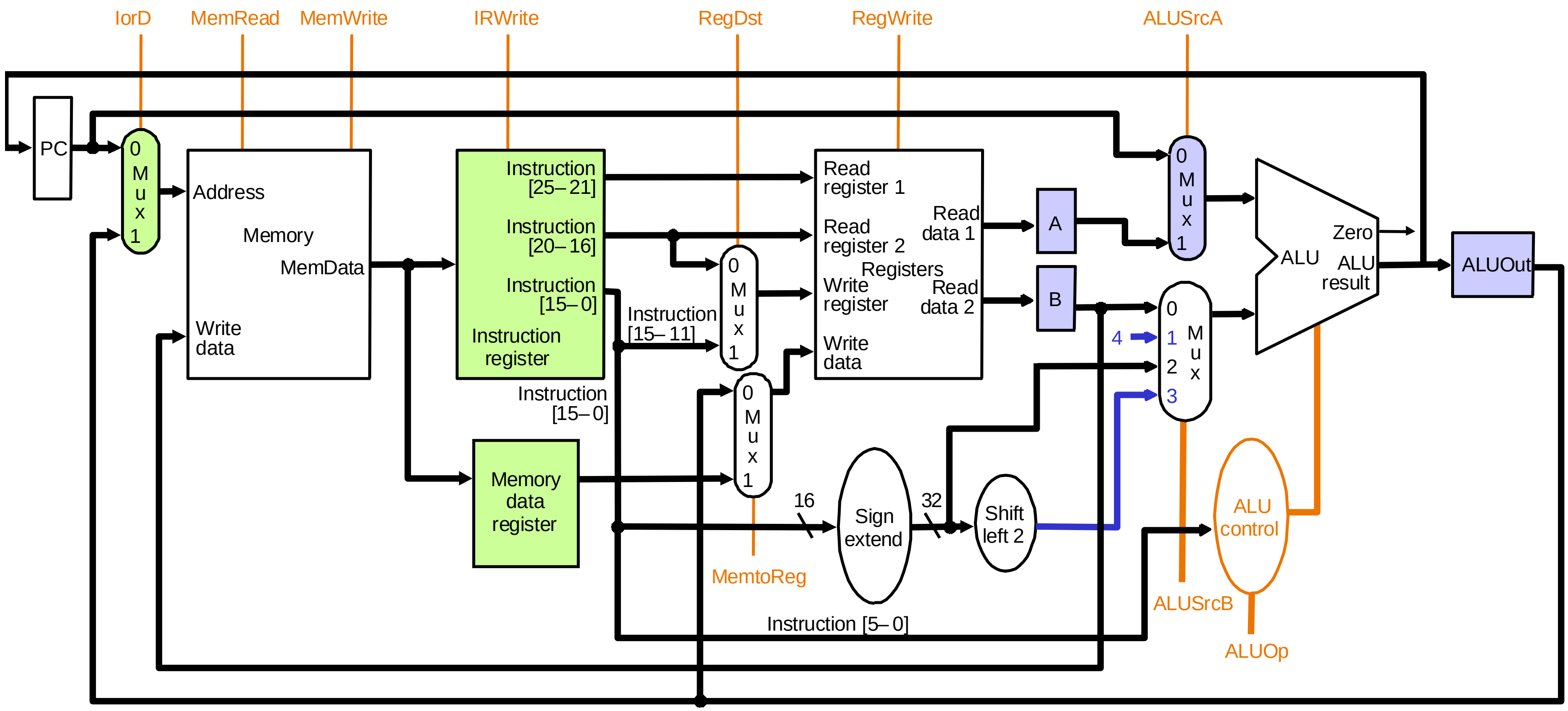


Harvard Architecture: Separate instruction and data memory

Review: Multi vs. Single-cycle Processor Datapath

Combine adders: add 1½ Mux & 3 temp. registers, A, B, ALUOut

Combine Memory: add 1 Mux & 2 temp. registers, IR, MDR



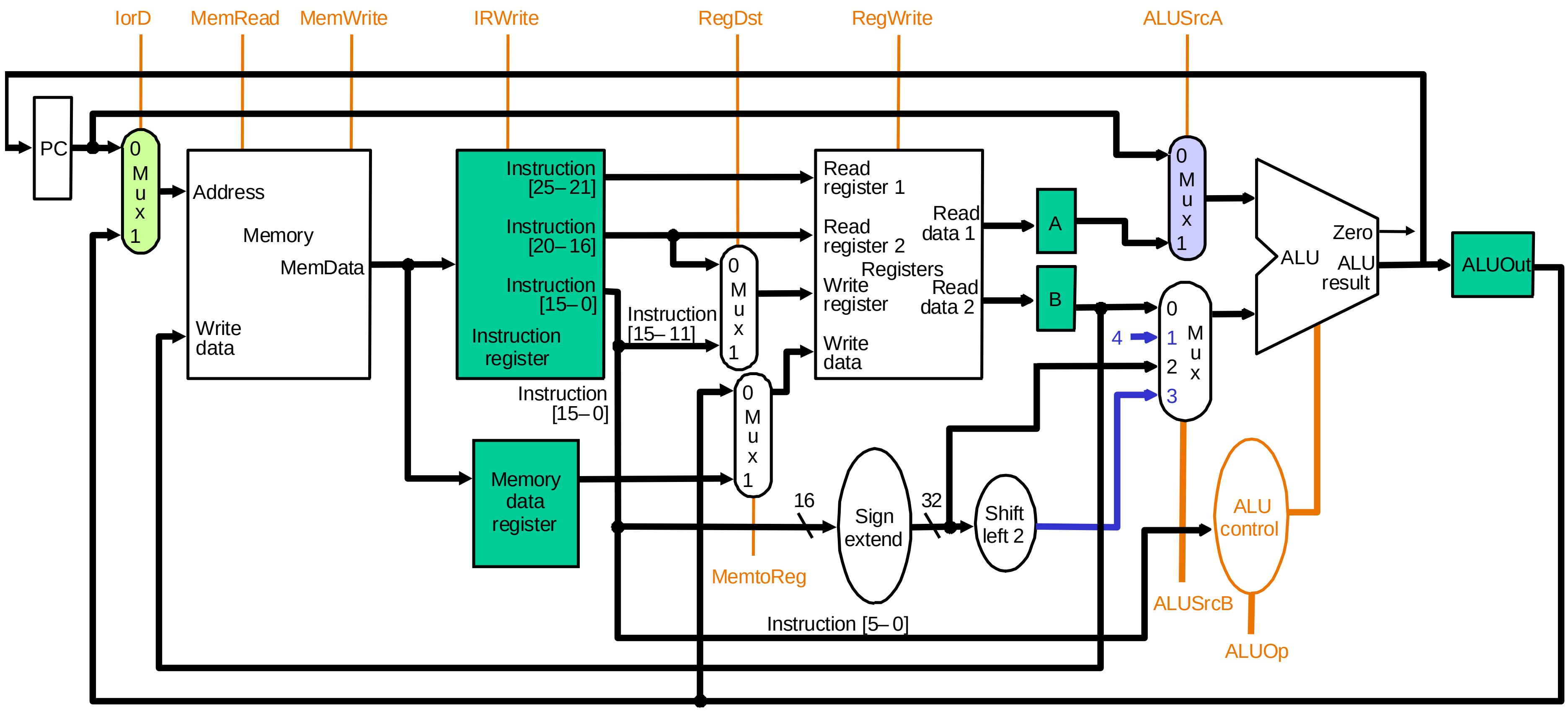
Single-cycle= 1 ALU + 2 Mem + 4 Muxes + 2 adders + OpcodeDecoders

Multi-cycle = 1 ALU + 1 Mem + 5½ Muxes + 5 Reg (IR,A,B,MDR,ALUOut) + FSM

Review: Multi-cycle Processor Datapath

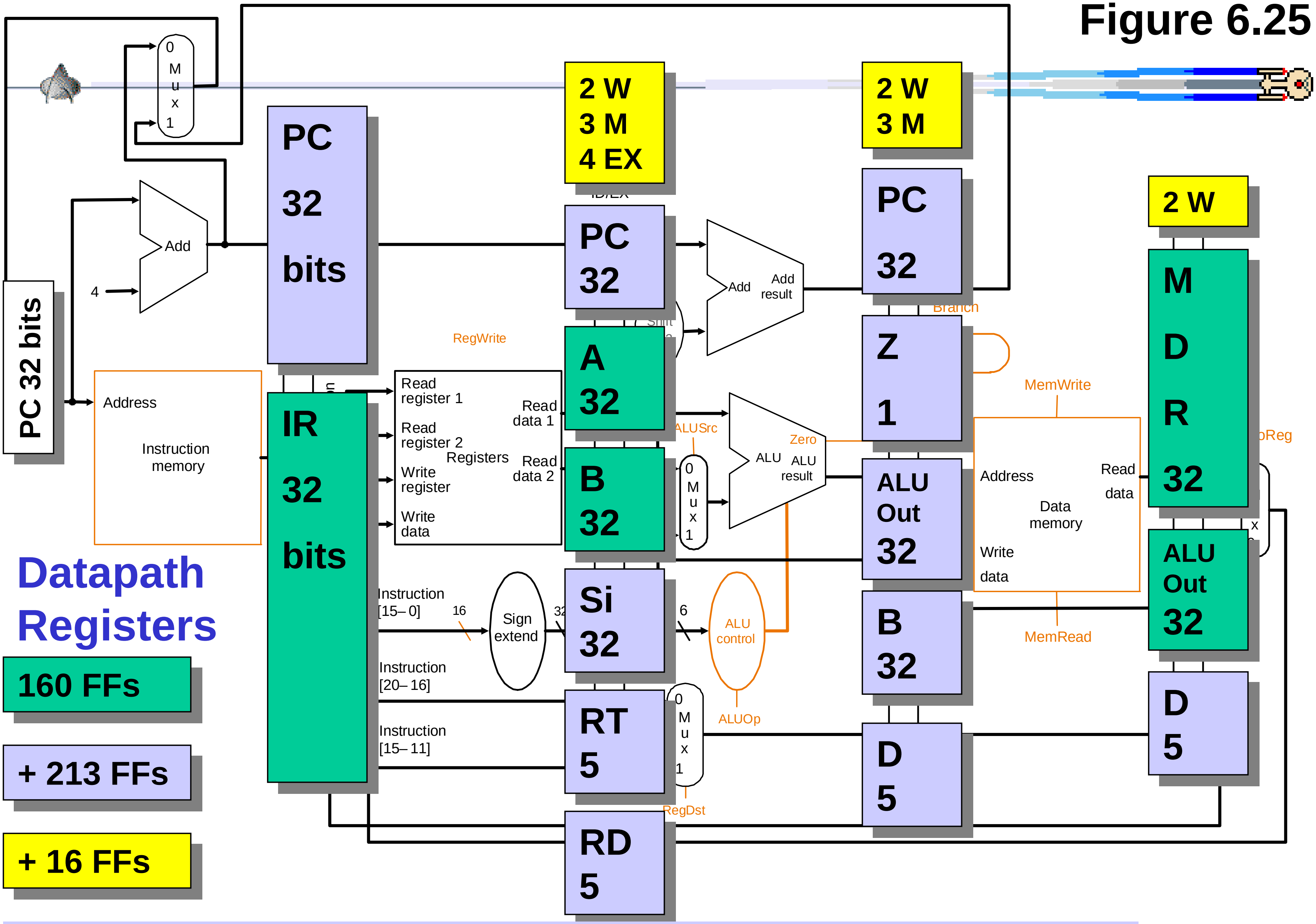
Single-cycle= 1 ALU + 2 Mem + 4 Muxes + 2 adders + OpcodeDecoders

Multi-cycle = 1 ALU + 1 Mem + 5½ Muxes + 5 Reg (IR,A,B,MDR,ALUOut) + FSM



5x32 = 160 additional FFs for multi-cycle processor over single-cycle processor

Figure 6.25



Datapath Registers

160 FFs

+ 213 FFs

+ 16 FFs

213+16 = 229 additional FFs for pipeline over multi-cycle processor

Review: Overhead

Chip Area Speed

Single-cycle model

- 8 ns Clock (125 MHz), (non-overlapping)
- 1 ALU + 2 adders
- 0 Muxes
- 0 Datapath Register bits (Flip-Flops)

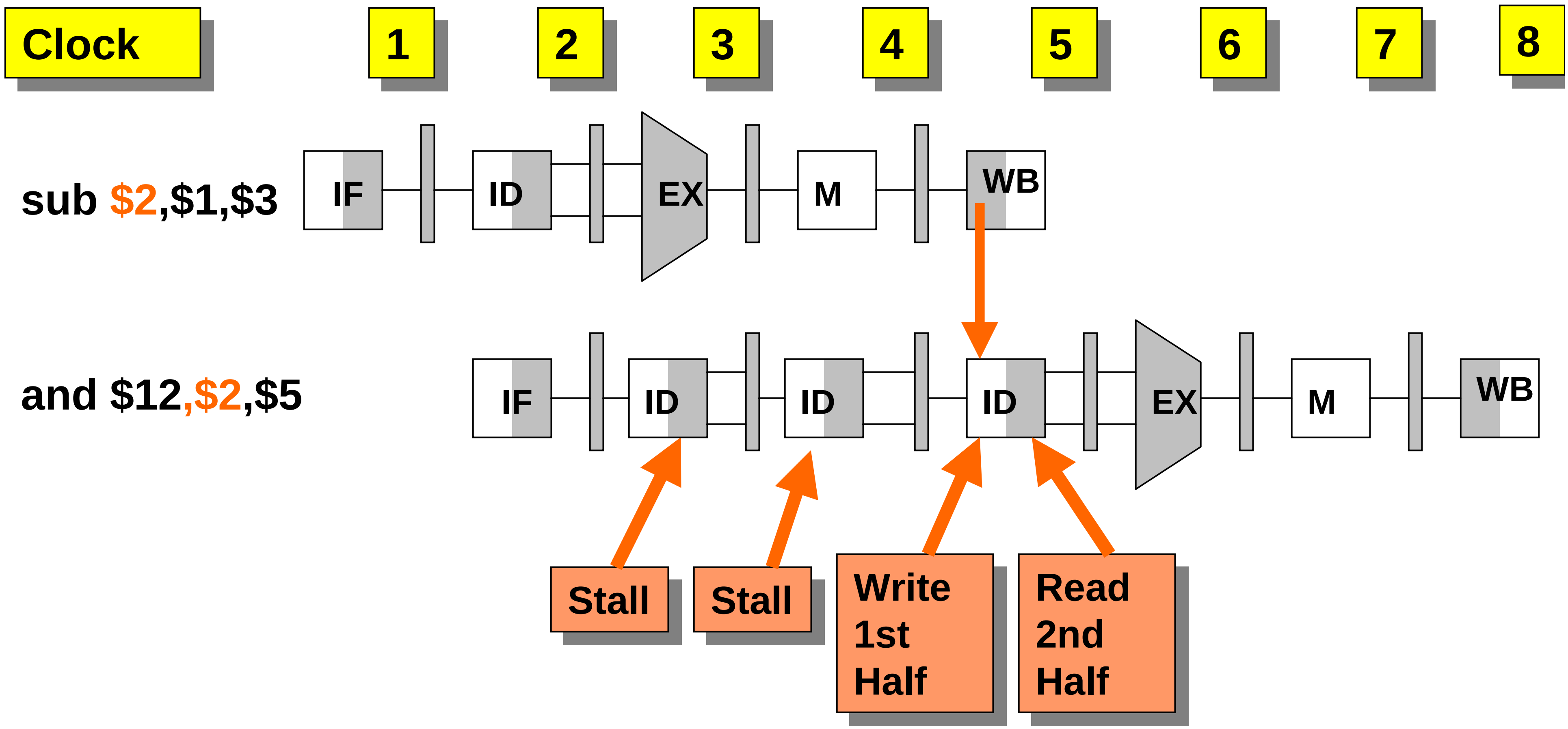
Multi-cycle model

- 2 ns Clock (500 MHz), (non-overlapping)
- 1 ALU + Controller
- 5 Muxes
- 160 Datapath Register bits (Flip-Flops)

Pipeline model

- 2 ns Clock (500 MHz), (overlapping)
- 2 ALU + Controller
- 4 Muxes
- 373 Datapath + 16 Controlpath Register bits (Flip-Flops)

Review: Data Dependencies: no forwarding



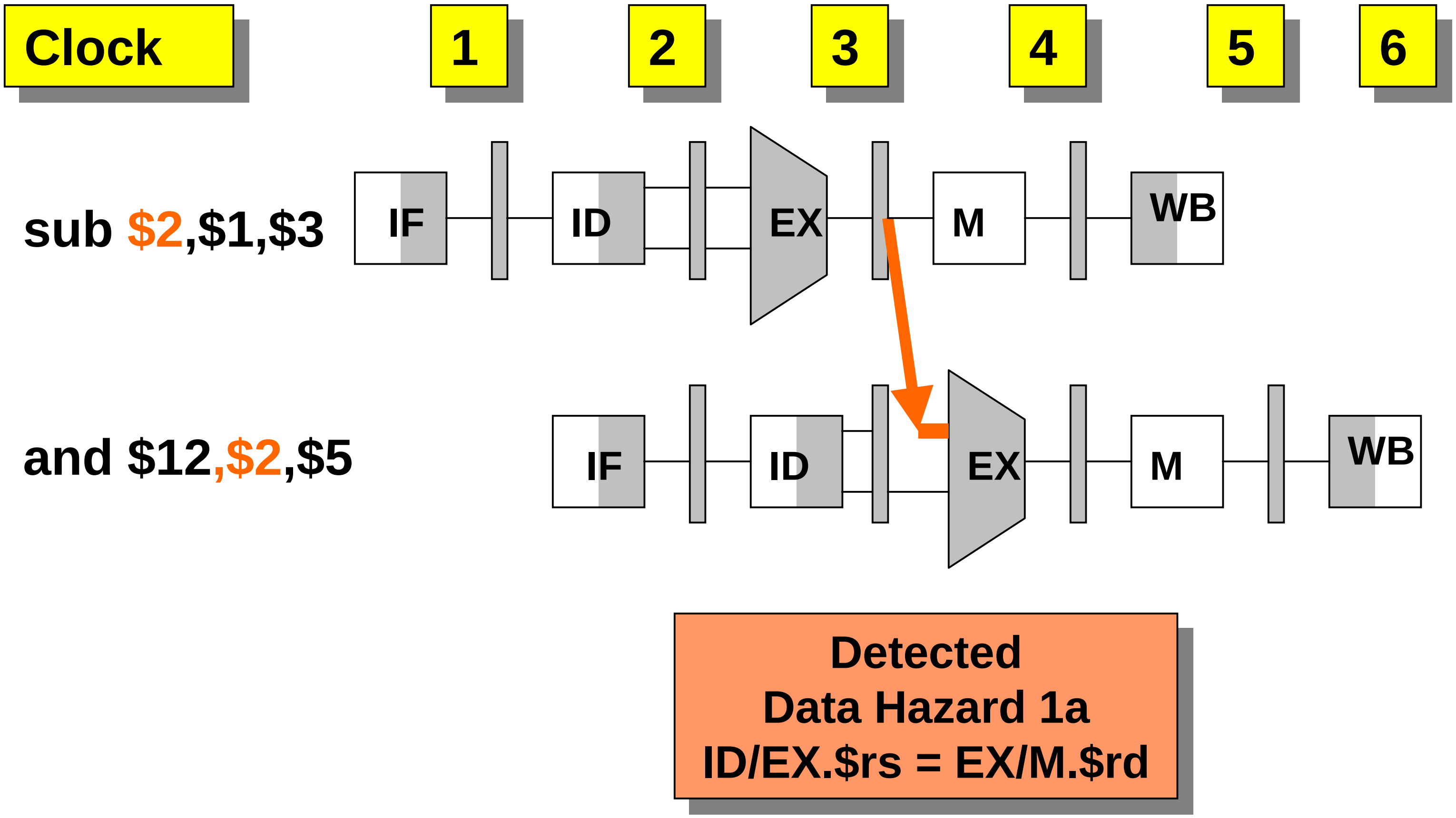
Suppose every instruction is dependant = 1 + 2 stalls = 3 clocks

$$\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500\ \text{Mhz}}{3} = 167\ \text{MIPS}$$

Review: R-Format Data Dependencies: Hazard Conditions

1a Data Hazard (2 stalls):				**EX/MEM.$rd = ID/EX.$rs**			
sub	$2,	$1,	$3	sub	$rd,	$rs,	$rt
and	$12,	$2,	$5	and	$rd,	$rs,	$rt
1b Data Hazard (2 stalls):				**EX/MEM.$rd = ID/EX.$rt**			
sub	$2,	$1,	$3	sub	$rd,	$rs,	$rt
and	$12,	$1,	$2	and	$rd,	$rs,	$rt
2a Data Hazard (1 stall):				**MEM/WB.$rd = ID/EX.$rs**			
sub	$2,	$1,	$3	sub	$rd,	$rs,	$rt
and	$12,	$1,	$5	sub	$rd,	$rs,	$rt
or	$13,	$2,	$1	and	$rd,	$rs,	$rt
2b Data Hazard (1 stall):				**MEM/WB.$rd = ID/EX.$rt**			
sub	$2,	$1,	$3	sub	$rd,	$rs,	$rt
and	$12,	$1,	$5	sub	$rd,	$rs,	$rt
or	$13,	$6,	$2	and	$rd,	$rs,	$rt

Data Dependencies (hazard 1a and 1b): with forwarding



Can R-Format dependencies completely be resolved by forwarding?

and	**$12,$2,$5**
beq	**$12,$0,L7**

Load Data Hazards: Hazard detection unit (page 490)

Stall Condition

Source	Destination
IF/ID.$rs IF/ID.$rt	= ID/EX.$rt $\wedge$ ID/EX.MemRead=1

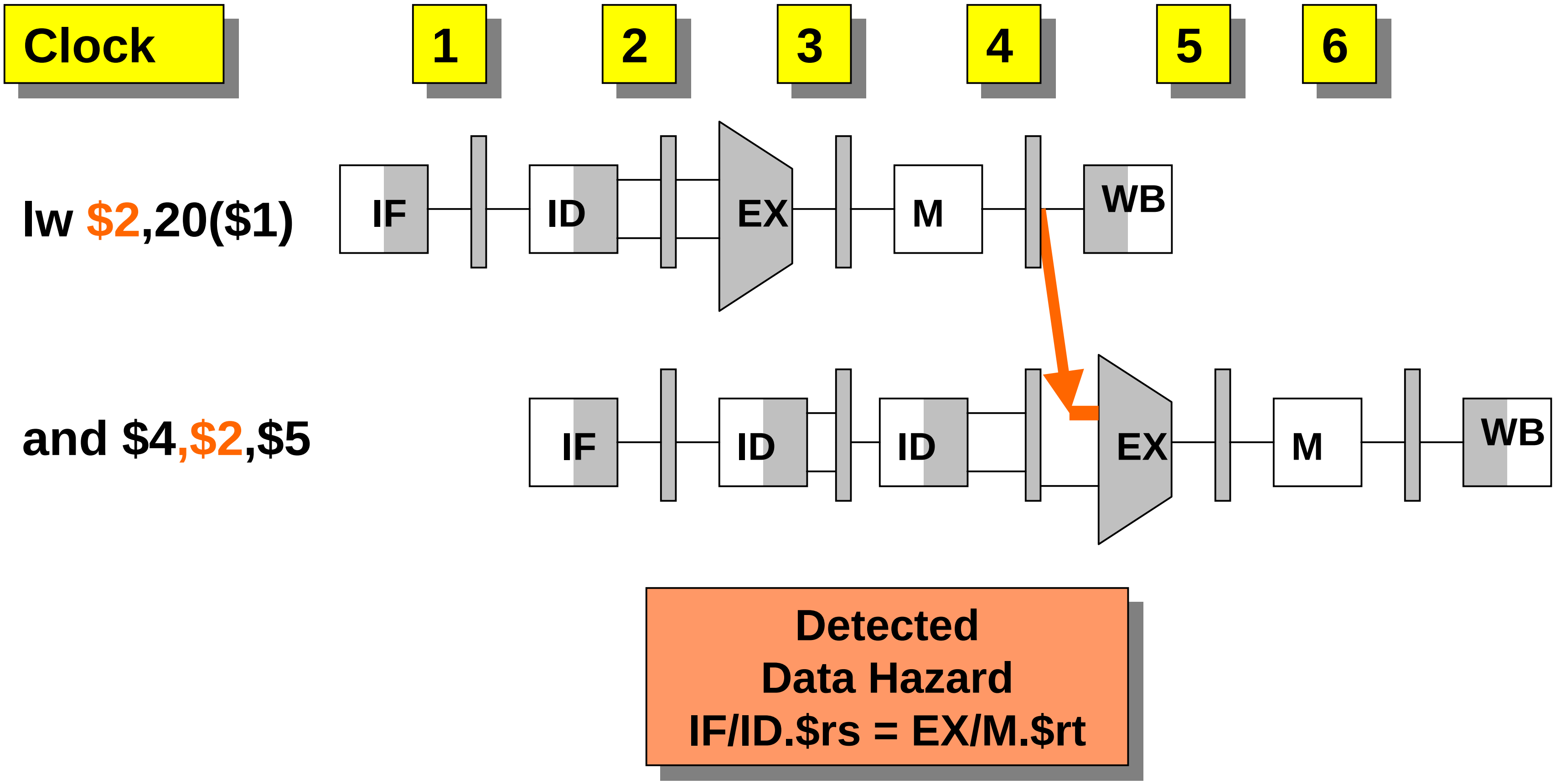
Stall Example

lw	$2,	20($1)		lw	$rt,	addr($rs)	
and	$4,	$2,	$5	and	$rd,	$rs,	$rt

No Stall Example: (only need to look at next instruction)

lw	$2,	20($1)		lw	$rt,	addr($rs)	
and	$4,	$1,	$5	and	$rd,	$rs,	$rt
or	$8,	$2,	$6	or	$rd,	$rs,	$rt

Load Data Dependencies: with forwarding



Load dependencies cannot be completely resolved by forwarding

Even through the Load stalls the next instruction, the stall time is added to the load instruction and not the next instruction.

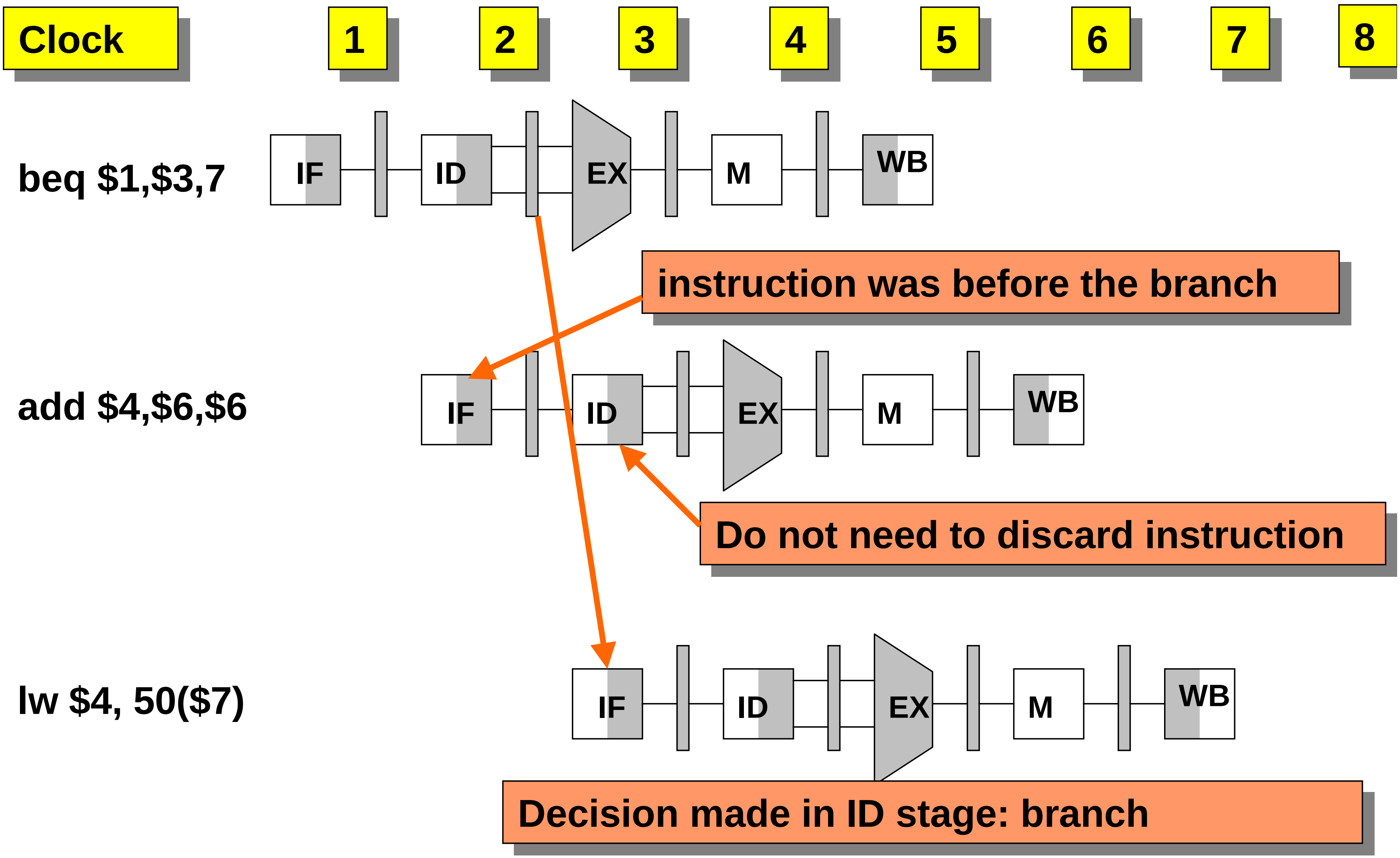
Load time = 1 (no dependancy) to 2 (with dependency on next instruction)

Delay slot

Before	After
add $4,$6,$6	
beq $1,$3,L7	beq $1,$3,7
	add $4,$6,$6
• • •	• • •
L7: lw $4, 50($7)	L7: lw $4, 50($7)

Can you move the add instruction into the delay slot?	No - but a delay slot still requires an instruction
add $4,$6,$6	add $4,$6,$6
beq $1,$4,L7	beq $1,$4,L7
	add $0,$0,$0

Branch Hazards: Soln #3, Delayed Decision



Summary: Instruction Hazards

	No-Forwarding	Forwarding	Hazard
R-Format	1-3	1	Data
Load	1-3	1-2	Data, Structural
Store	1	1-2	Structural

	No Delay Slot	Delay Slot	Hazard
Branch (decision is made in the ID stage)	2	1	Control
Branch (decision is made in the EX stage)	3	1	Control
Jump	2	1	

Structural Hazard: **Instruction & Data memory combined.**

Performance, page 504

Also known as the instruction latency with in a pipeline

Pipeline throughput

Instruction	Single-Cycle	Multi-Cycle Clocks	Pipeline Cycles	Instruction Mix
loads	1	5	1.5 (50% dependancy)	23%
stores	1	4	1	13%
arithmetic	1	4	1	43%
branches	1	3	1.25 (25% dependancy)	19%
jumps	1	3	2	2%
Clock speed	125 Mhz 8 ns	500 Mhz 2 ns	500 Mhz 2 ns	
CPI	1	4.02	1.18	= Σ Cycles*Mix
MIPS	125 MIPS	125 MIPS	424 MIPS	= Clock/CPI

load instruction time = 50%*(1 clock) + 50%*(2 clocks)=1.5

branch time = 75%*(1 clocks) + 25%*(2 clocks) = 1.25

Pipelining and the cache (Designing…,M.J.Quinn, '87)

Instruction Pipelining is the use of pipelining to allow more than one instruction to be in some stage of execution at the same time.

Ferranti ATLAS (1963):

- **Pipelining reduced the average time per instruction by 375%**
- **Memory could not keep up with the CPU, needed a cache.**

Cache memory is a small, fast memory unit used as a buffer between a processor and primary memory

Principle of Locality

- **Principle of Locality**

  states that programs access a relatively small portion of their address space at any instance of time

- **Two types of locality**

  - **Temporal locality (locality in time)**

    If an item is referenced, then

    the same item will tend to be referenced soon

    "the tendency to reuse recently accessed data items"

  - **Spatial locality (locality in space)**

    If an item is referenced, then

    nearby items will be referenced soon

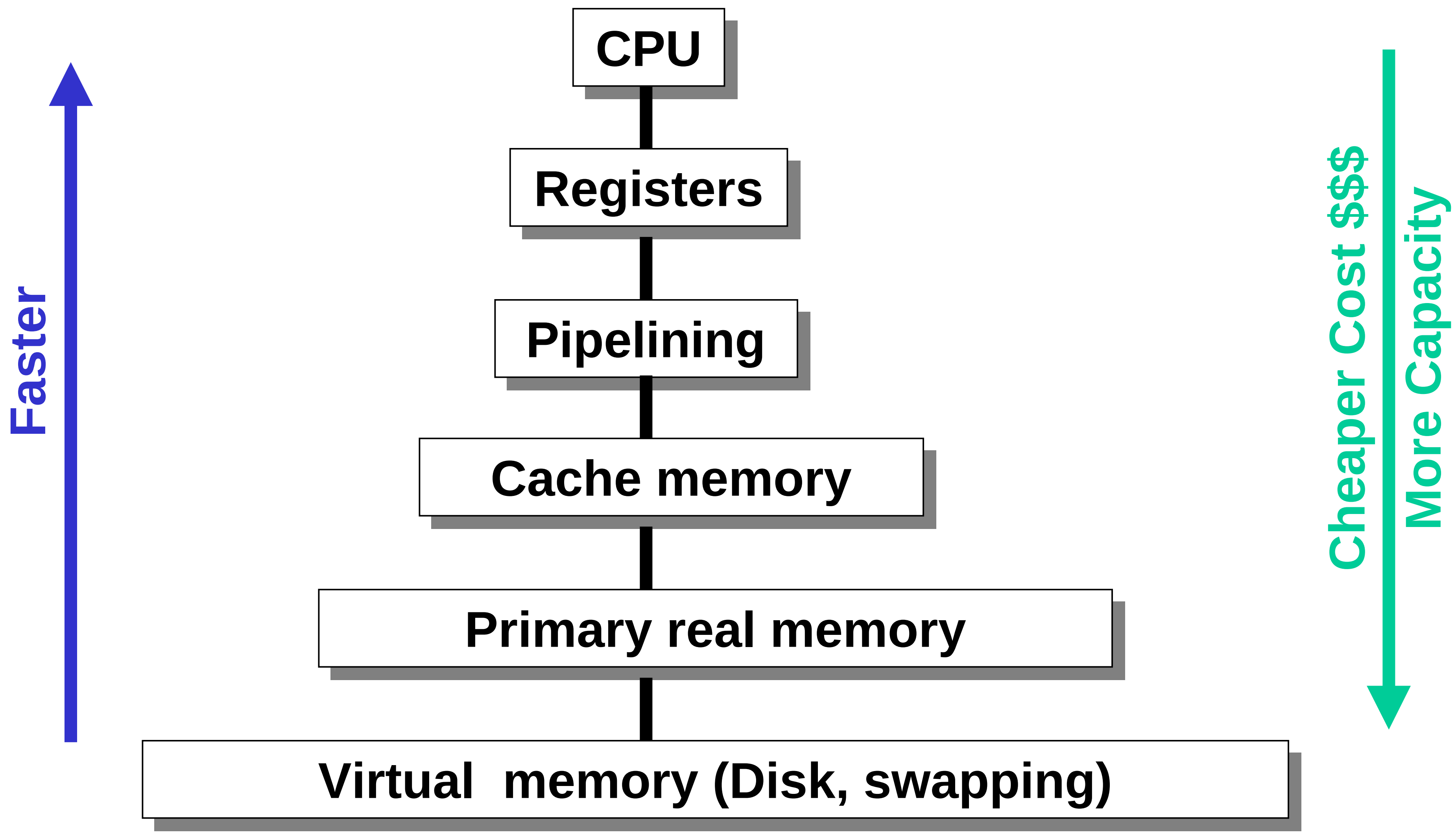
    "the tendency to reference nearby data items"

Memories Technology and Principle of Locality

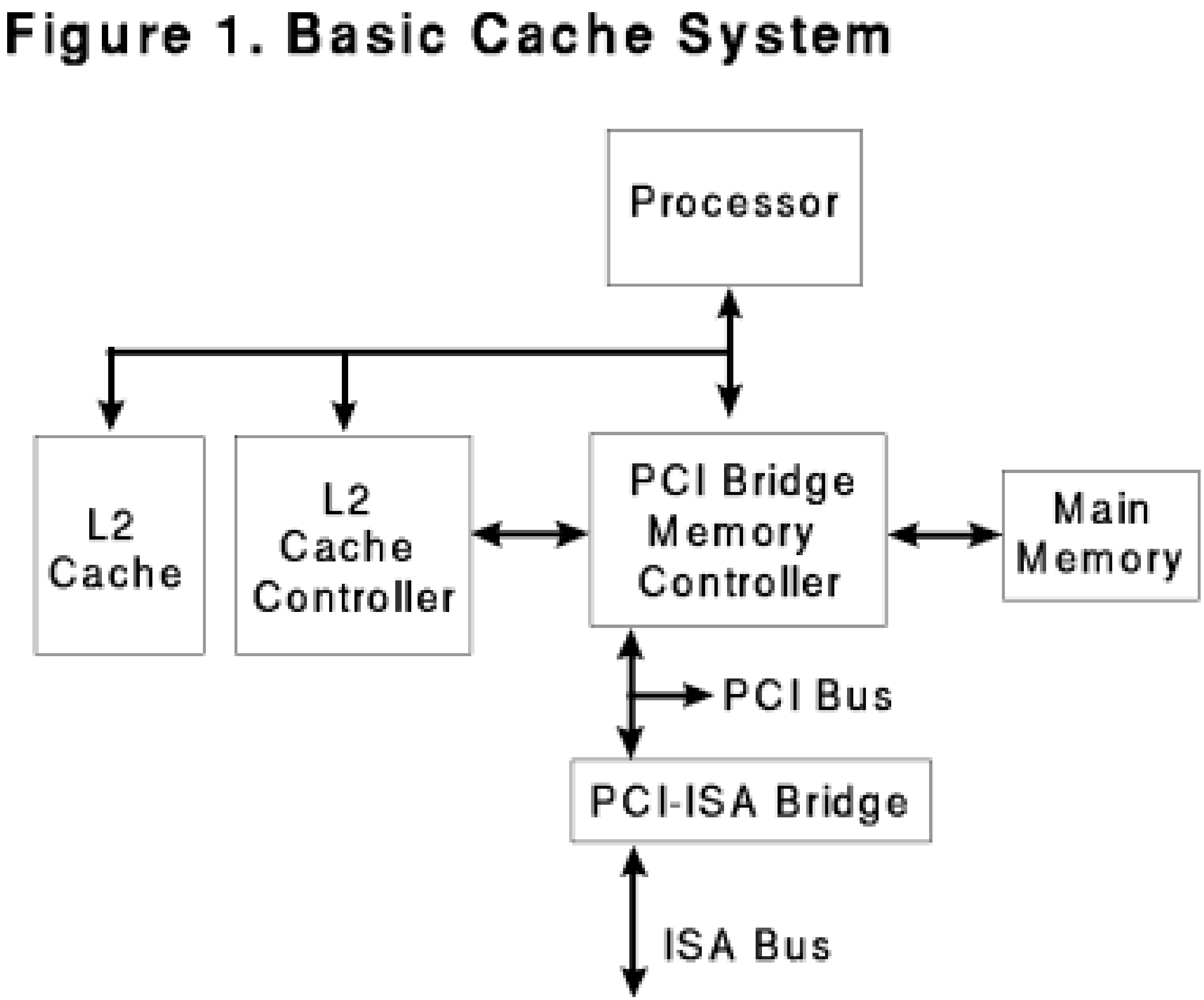
- Faster Memories are more expensive per bit
- Slower Memories are usually smaller in area size per bit

Memory Technology	Typical access time	$ per Mbyte
SRAM	5-25 ns	$100-$250
DRAM	60-120 ns	$5-$10
Magnetic Disk	10-20 million ns	$0.10-$0.20

Memory Hierarchy



Basic Cache System



Cache Terminology

A hit if the data requested by the CPU is in the upper level

Hit rate or **Hit ratio**

is the fraction of accesses found in the upper level

Hit time

is the time required to access data in the upper level

= <detection time for hit or miss> + <hit access time>

A miss if the data is not found in the upper level

Miss rate or **(1 – hit rate)**

is the fraction of accesses <u>not</u> found in the upper level

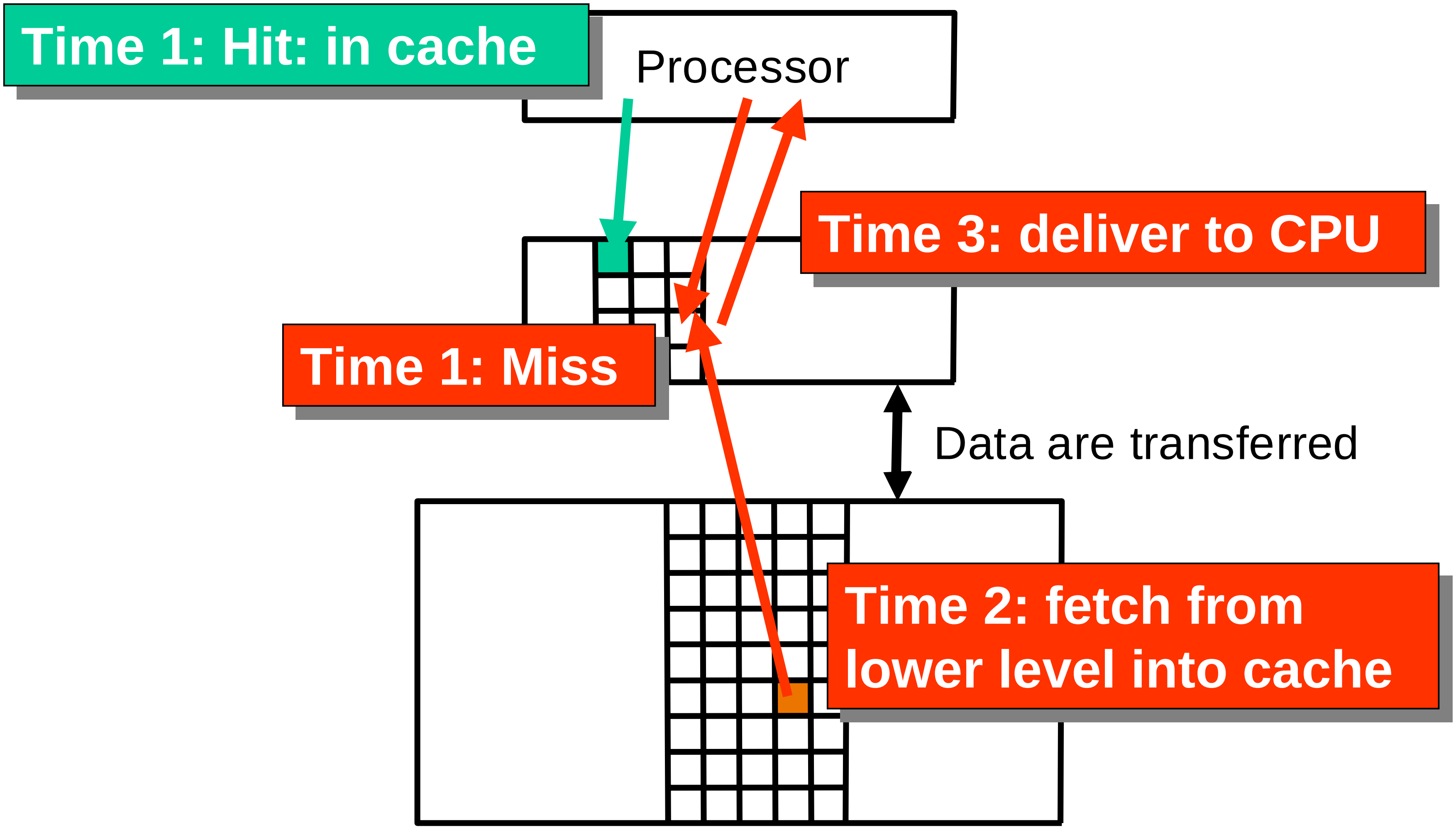
Miss penalty

is the time required to access data in the lower level

= <lower access time>+<reload processor time>

Cache Example

Figure 7.2



Hit time = Time 1

Miss penalty = Time 2 + Time 3

Cache Memory Technology: SRAM

see page B-27

- Why use SRAM (Static Random Access Memory)?

- Speed.

  The primary advantage of an SRAM over DRAM is speed.

  The fastest DRAMs on the market still require 5 to 10 processor clock cycles to access the first bit of data.

  SRAMs can operate at processor speeds of 250 MHz and beyond, with access and cycle times equal to the clock cycle used by the microprocessor

- Density.

  when 64 Mb DRAMs are rolling off the production lines, the largest SRAMs are expected to be only 16 Mb.

see reference: http://www.chips.ibm.com/products/memory/sramoperations/sramop.html

Cache Memory Technology: SRAM (con’t)

- **Volatility.**

  Unlike DRAMs, SRAM cells do not need to be refreshed. SRAMs are available 100% of the time for reading & writing.

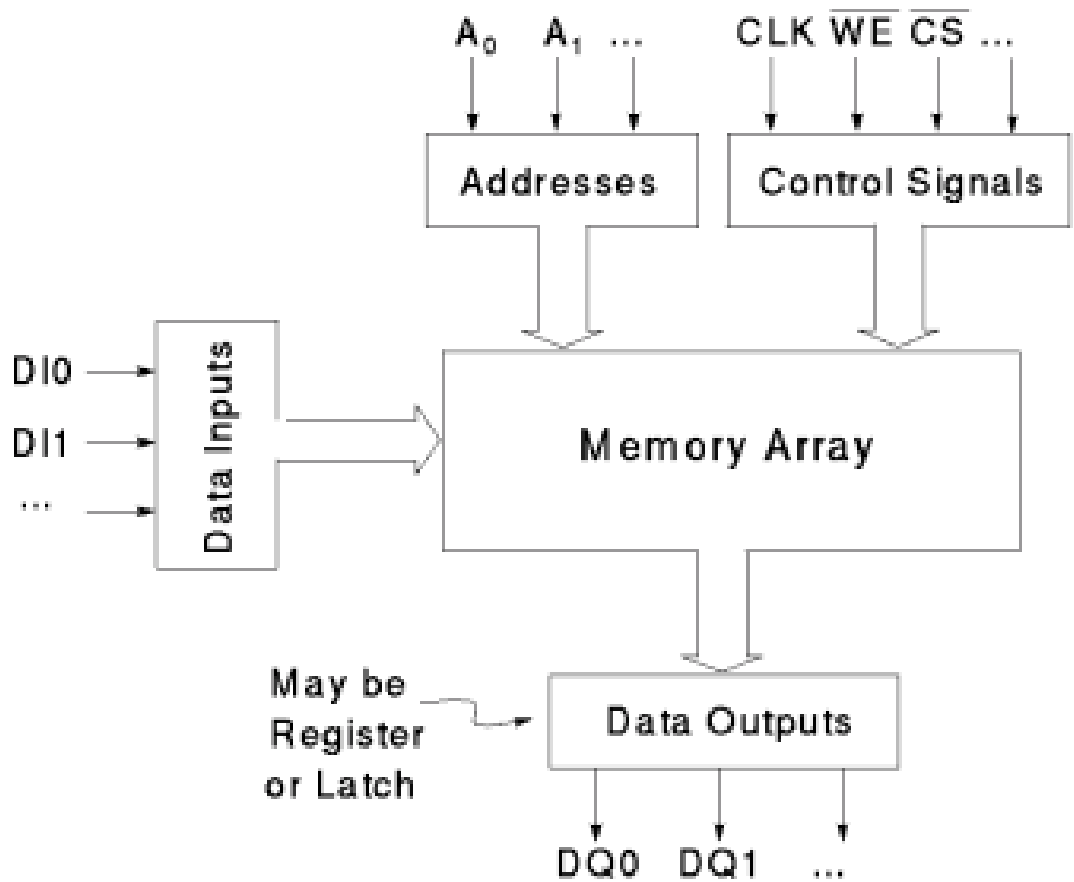
- **Cost.**

  If cost is the primary factor in a memory design, then DRAMs win hands down.

  If, on the other hand, performance is a critical factor, then a well-designed SRAM is an effective cost performance solution.

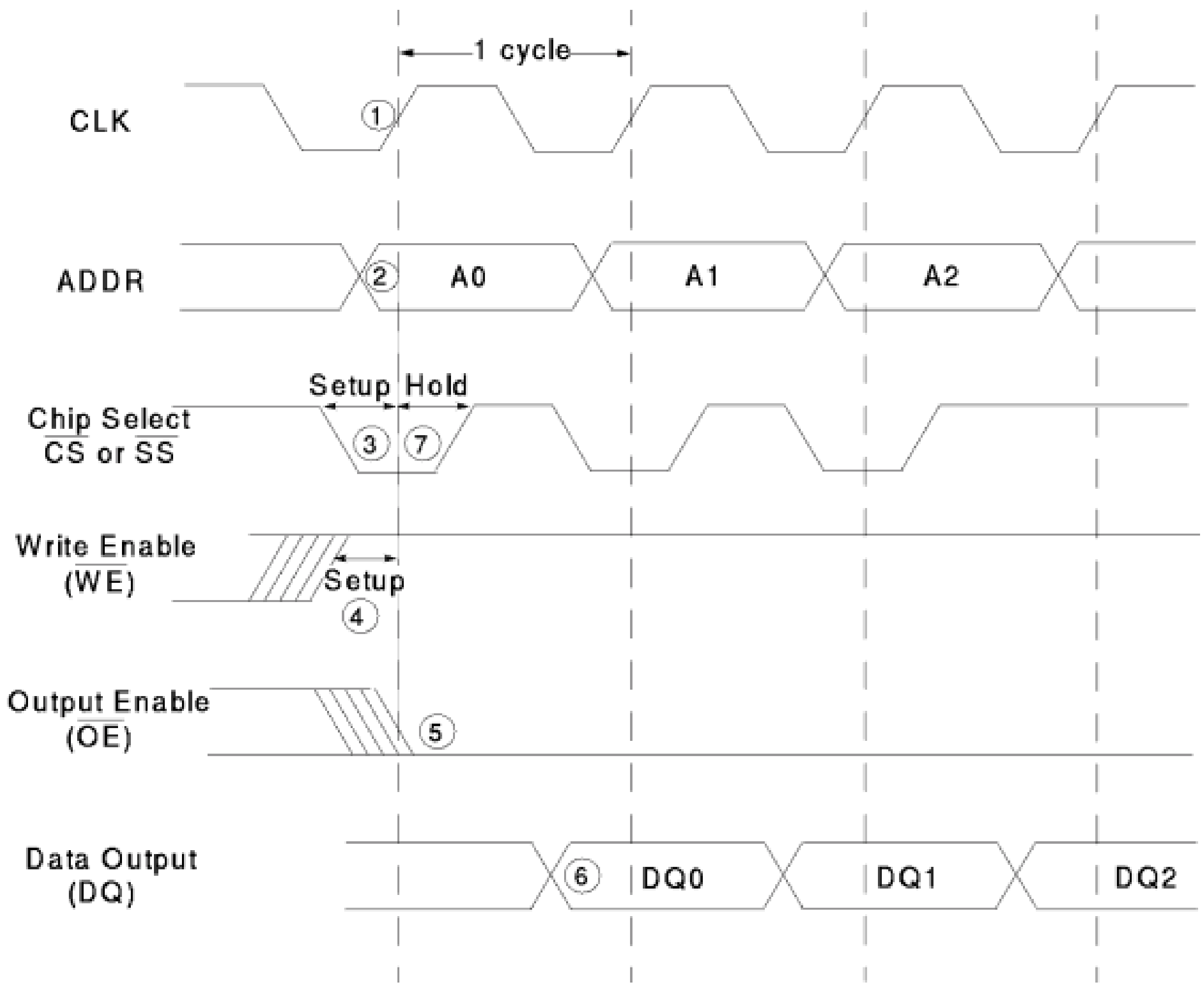
Cache Memory Technology: SRAM Block diagram

Figure 2. Simplified Block Diagram of a Synchronous SRAM



Cache Memory Technology: SRAM timing diagram

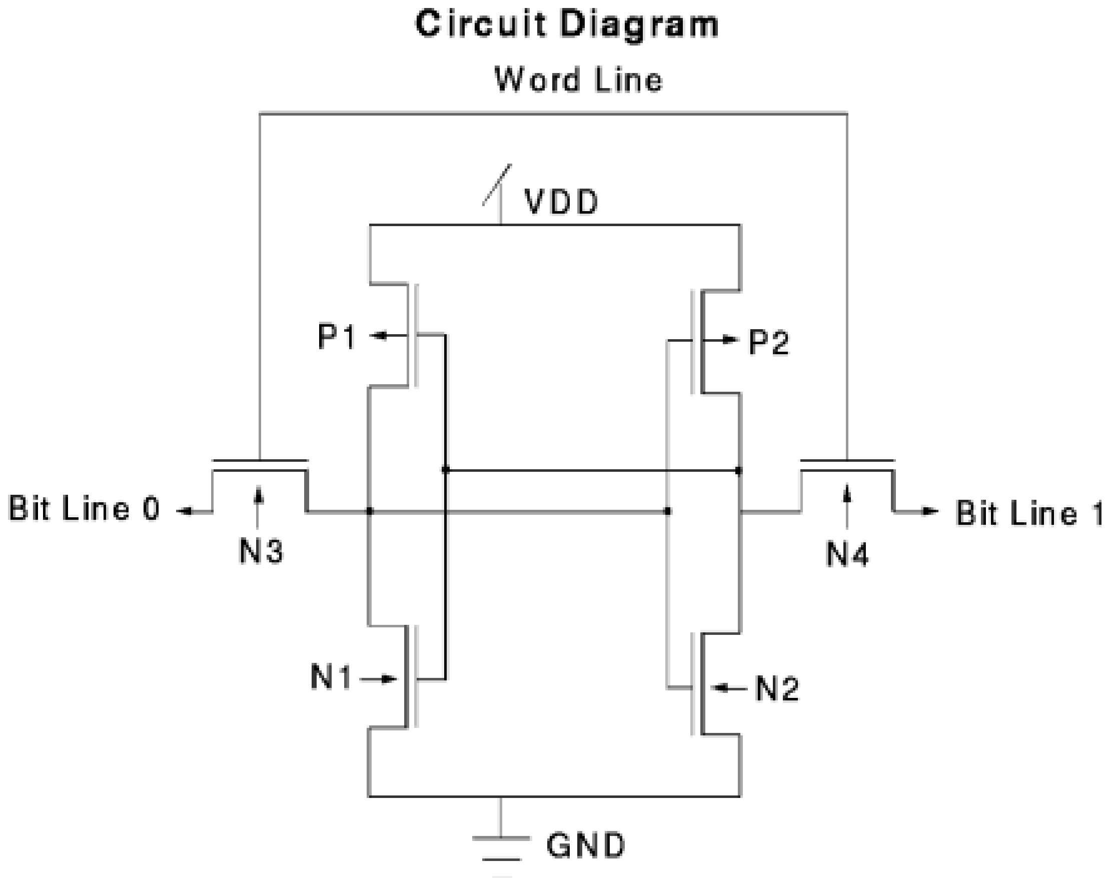
Figure 4. Reading from Memory (Flow Thru mode)

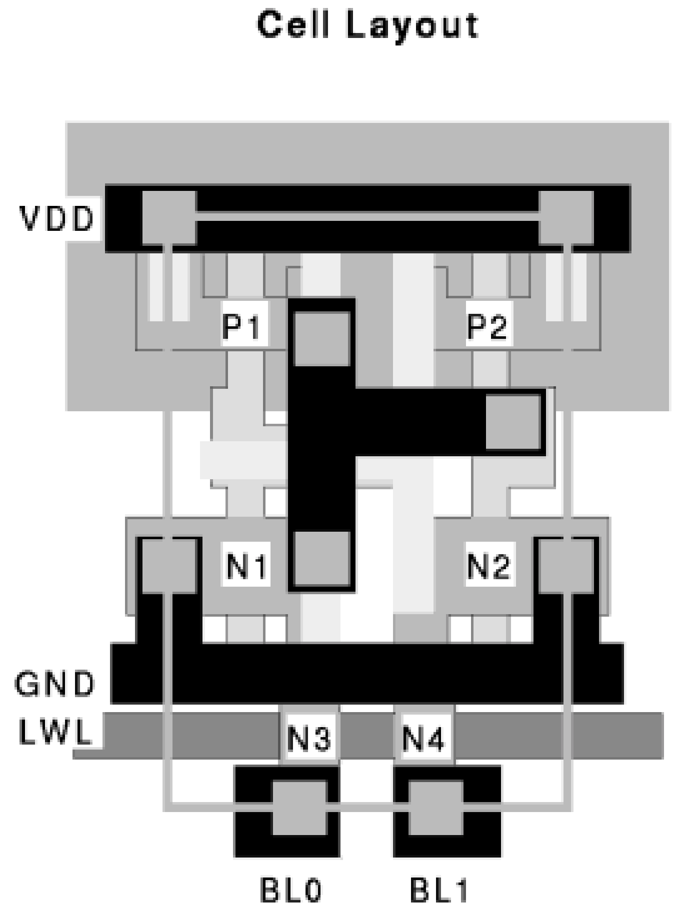


Note: DQ0 is the data associated with Address 0 (A0). DQ1 is the data associated with Address 1 (A1).

Cache Memory Technology: SRAM 1 bit cell layout

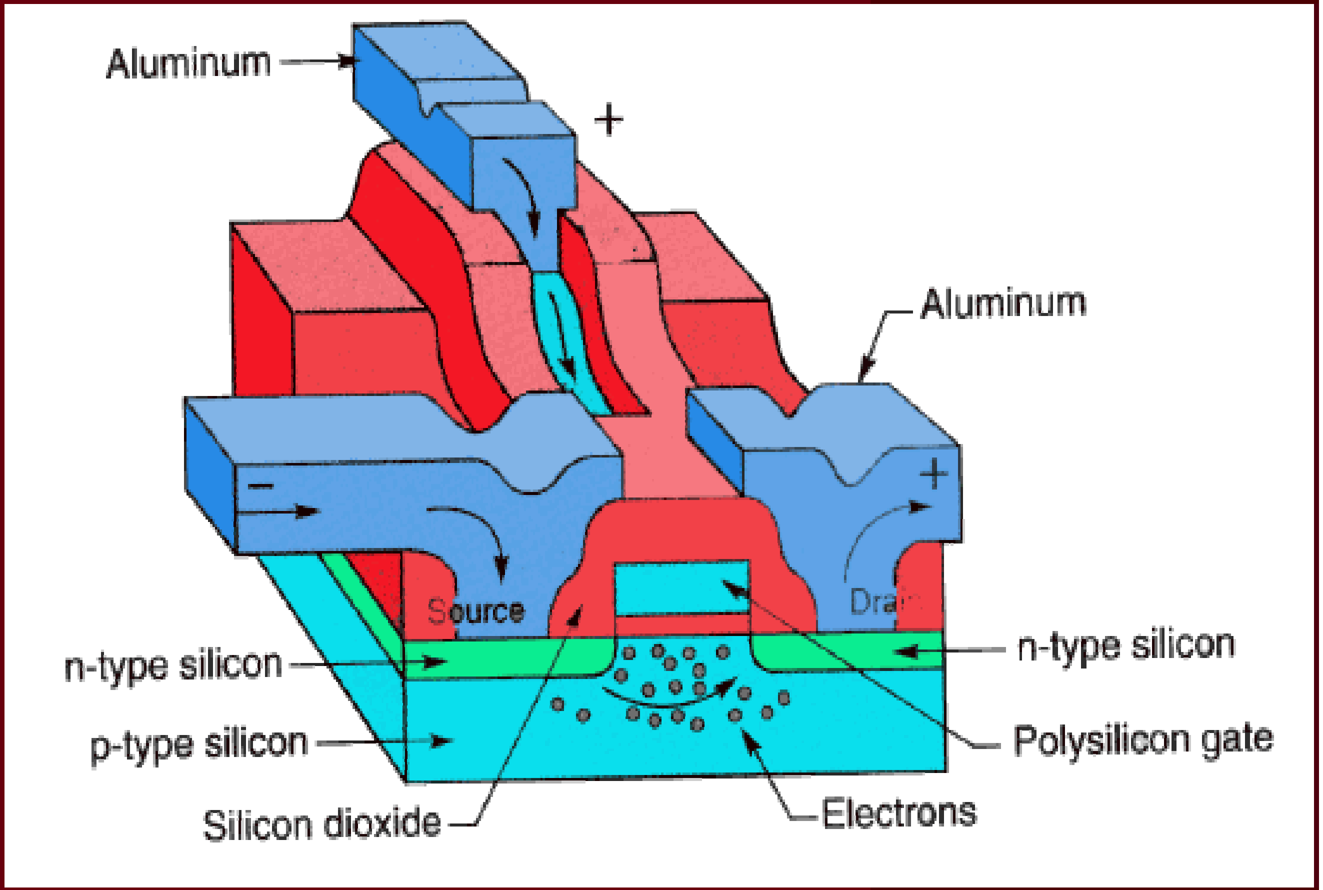
Figure 3. IBM's 6-Transistor Memory Cell





Real transistor

- 3-D structure
- Real materials

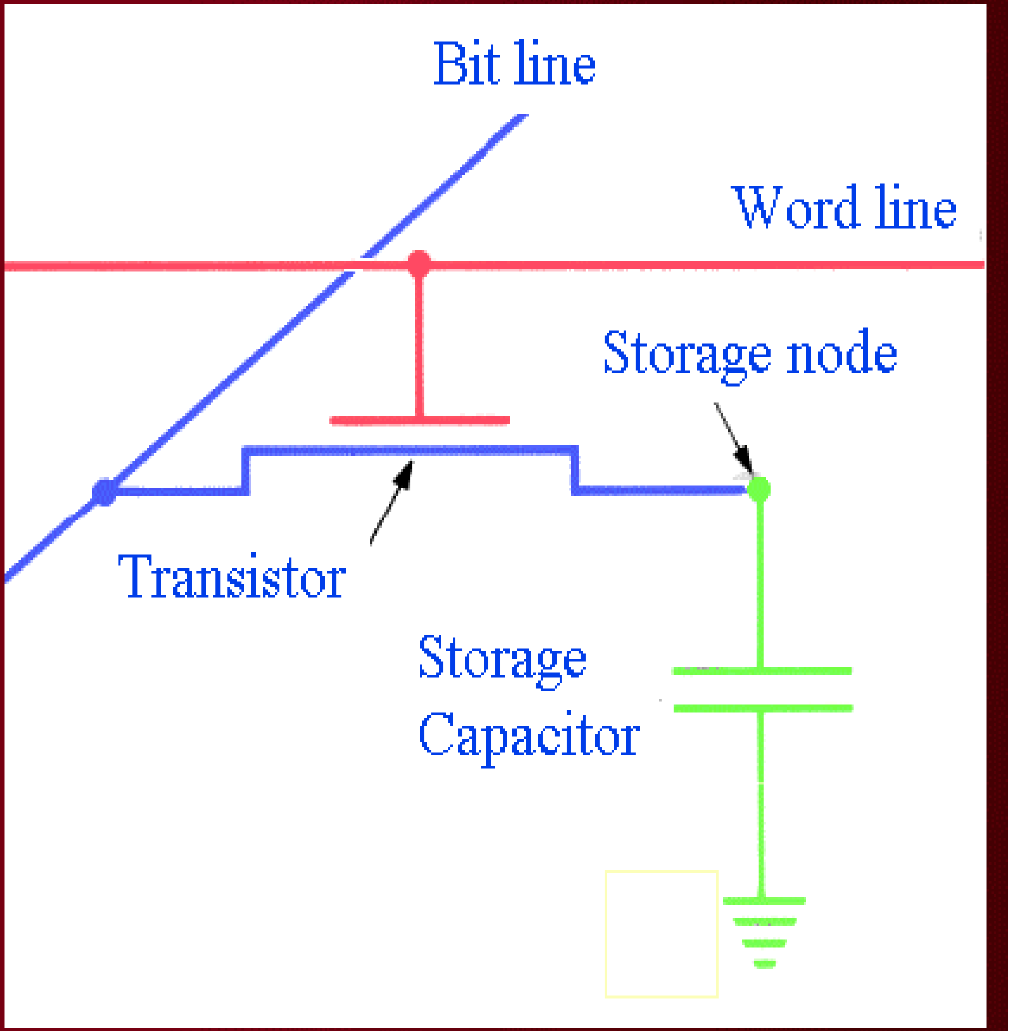


Ref: http://www.msm.cam.ac.uk/dmg/teaching/m101999/Ch8/index.htm

see page B-31

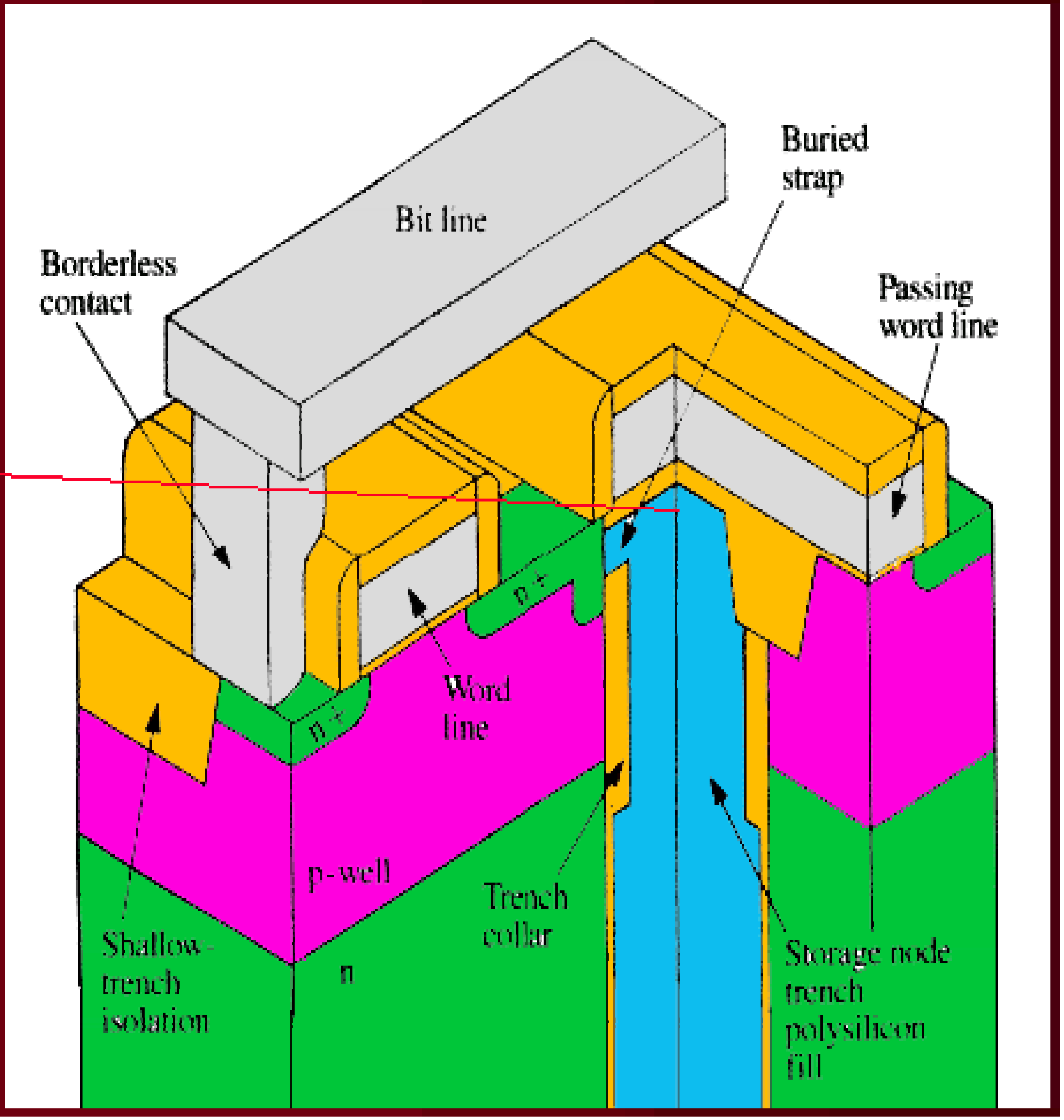
Basic DRAM design

- DRAM replaces all but one transitors of flip-flop with a capacitor
- => smaller!
- Capacitor stores information
- Charge leakage requires periodic refreshment (sense & rewrite)



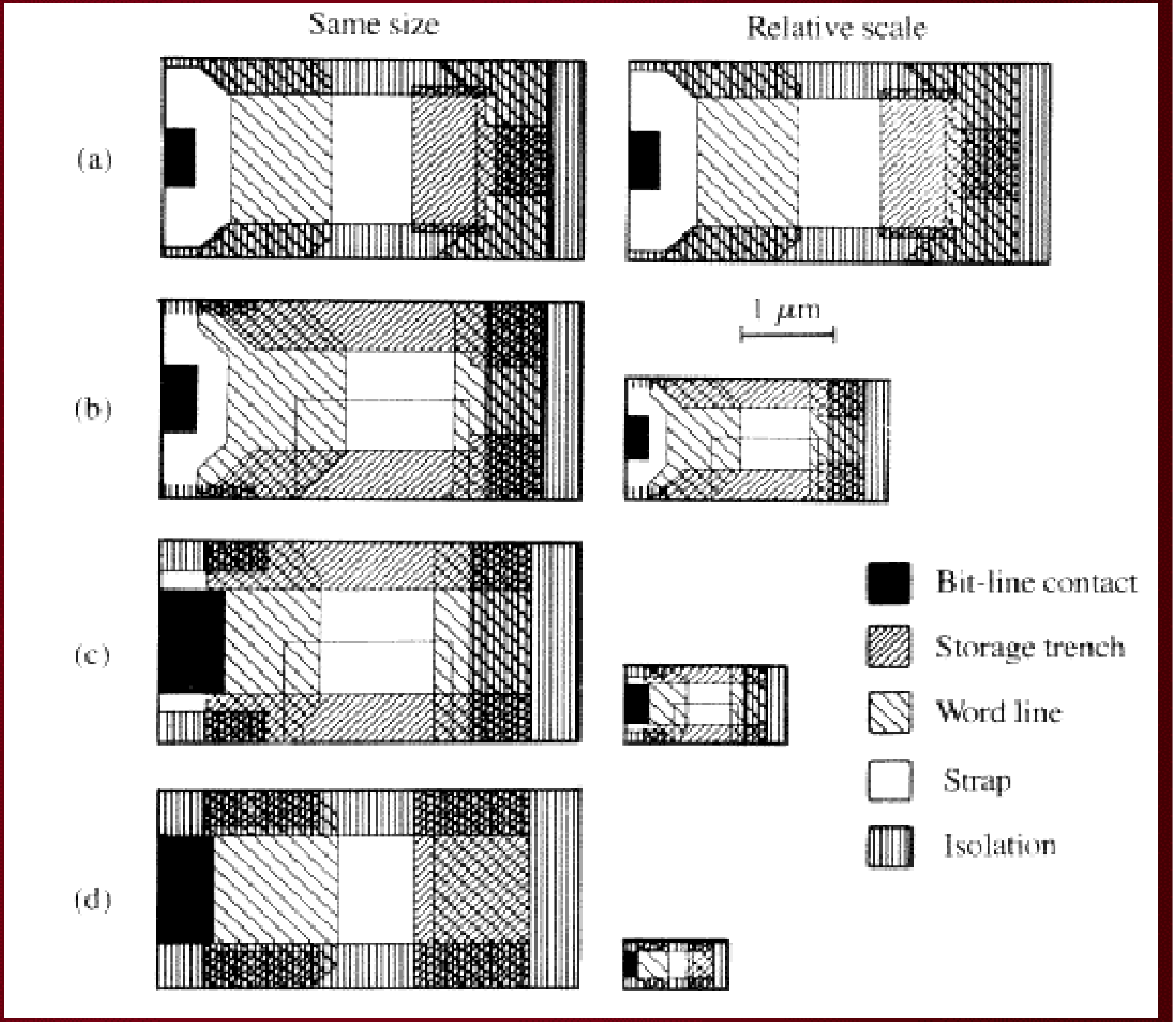
256Mb DRAM

- Increased vertical integration
- Word line passes over capacitor and contact
- Cell area ~0.5µm²
- Capacitor area smaller - dielectric must be thinner
- =>higher quality dielectric required

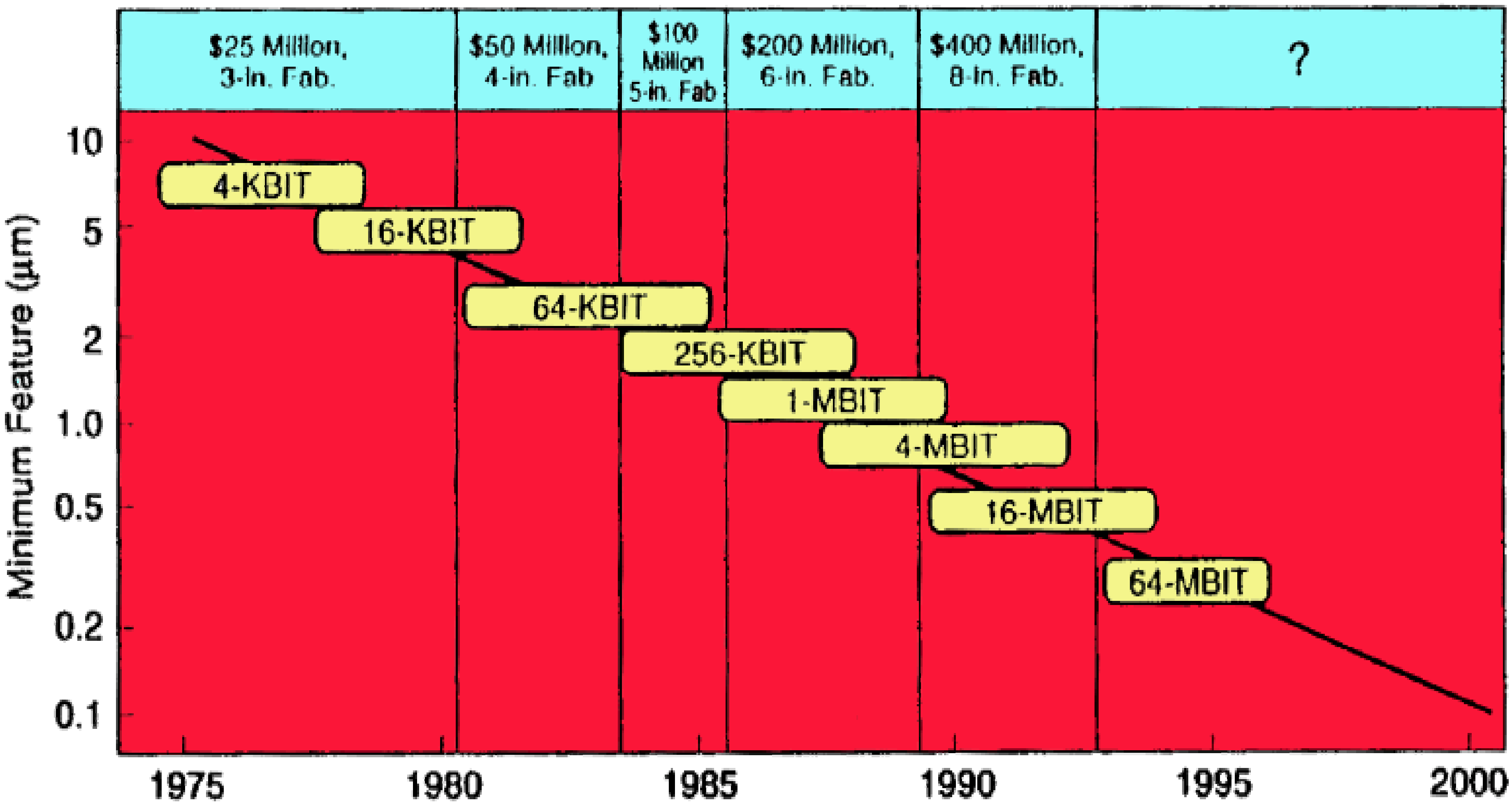


Memory Technology: DRAM Evolution

DRAM evolution (II)

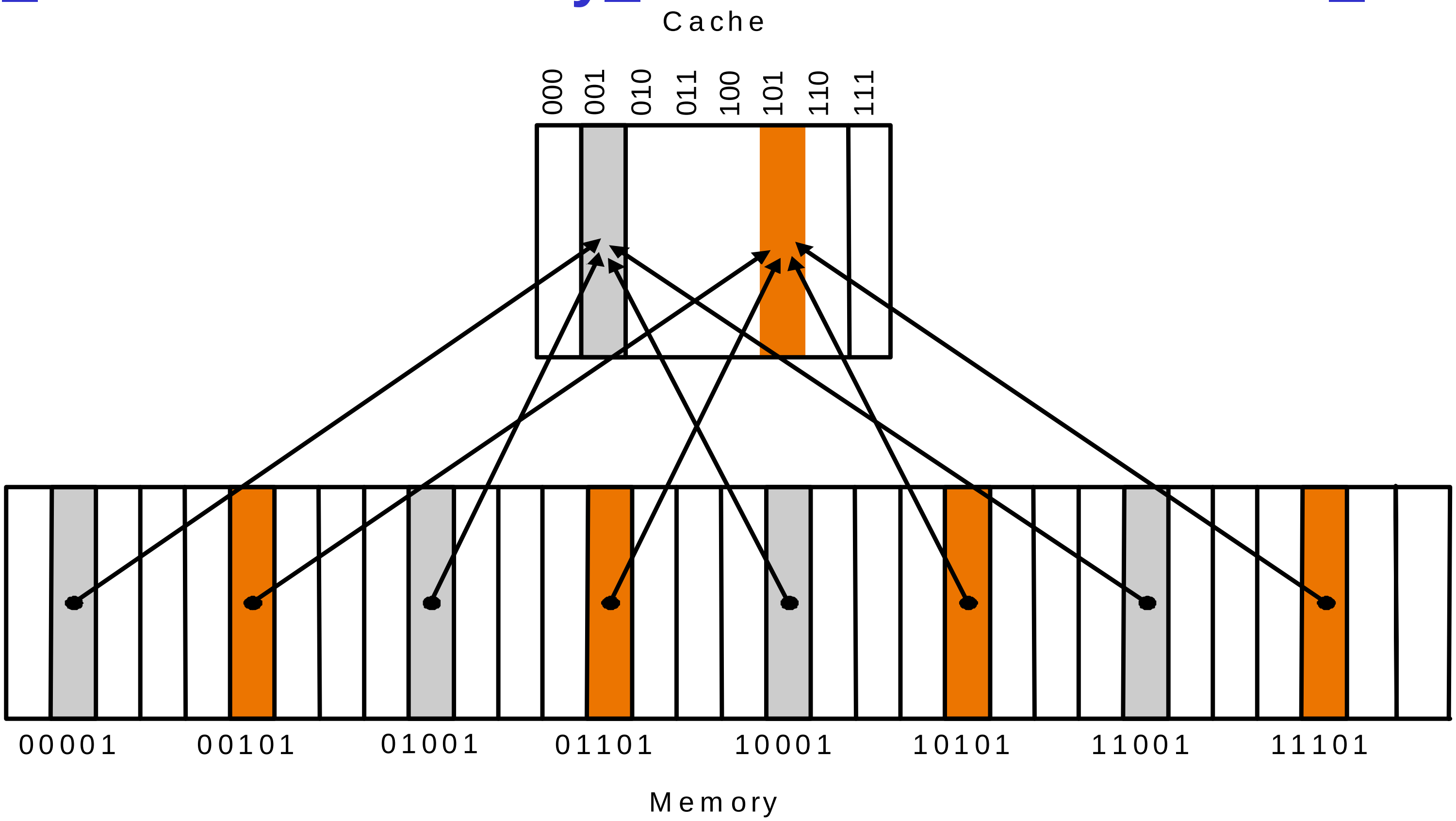


DRAM development



Direct Mapped Cache

- **Direct Mapped:** assign the cache location based on the address of the word in memory
- **cache_address = memory_address modulo cache_size;**



Observe there is a Many-to-1 memory to cache relationship

Direct Mapped Cache: Data Structure

There is a Many-to-1 relationship between memory and cache

How do we know whether the data in the cache corresponds to the requested word?

tags

- contain the address information required to identify whether a word in the cache corresponds to the requested word.
- tags need only to contain the upper portion of the memory address (often referred to as a page address)

valid bit

- indicates whether an entry contains a valid address

Direct Mapped Cache: Temporal Example

Figure 7.6

lw	$1,10 110 ($0)	Miss: valid	lw $1,22($0)
lw	$2,11 010 ($0)	Miss: valid	lw $2,26($0)
lw	$3,10 110 ($0)	Hit!	lw $3,22($0)

Index	Valid	Tag	Data
000	N		
001	N		
010	Y	11	Memory[11010]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110]
111	N		

Direct Mapped Cache: Worst case, always miss!

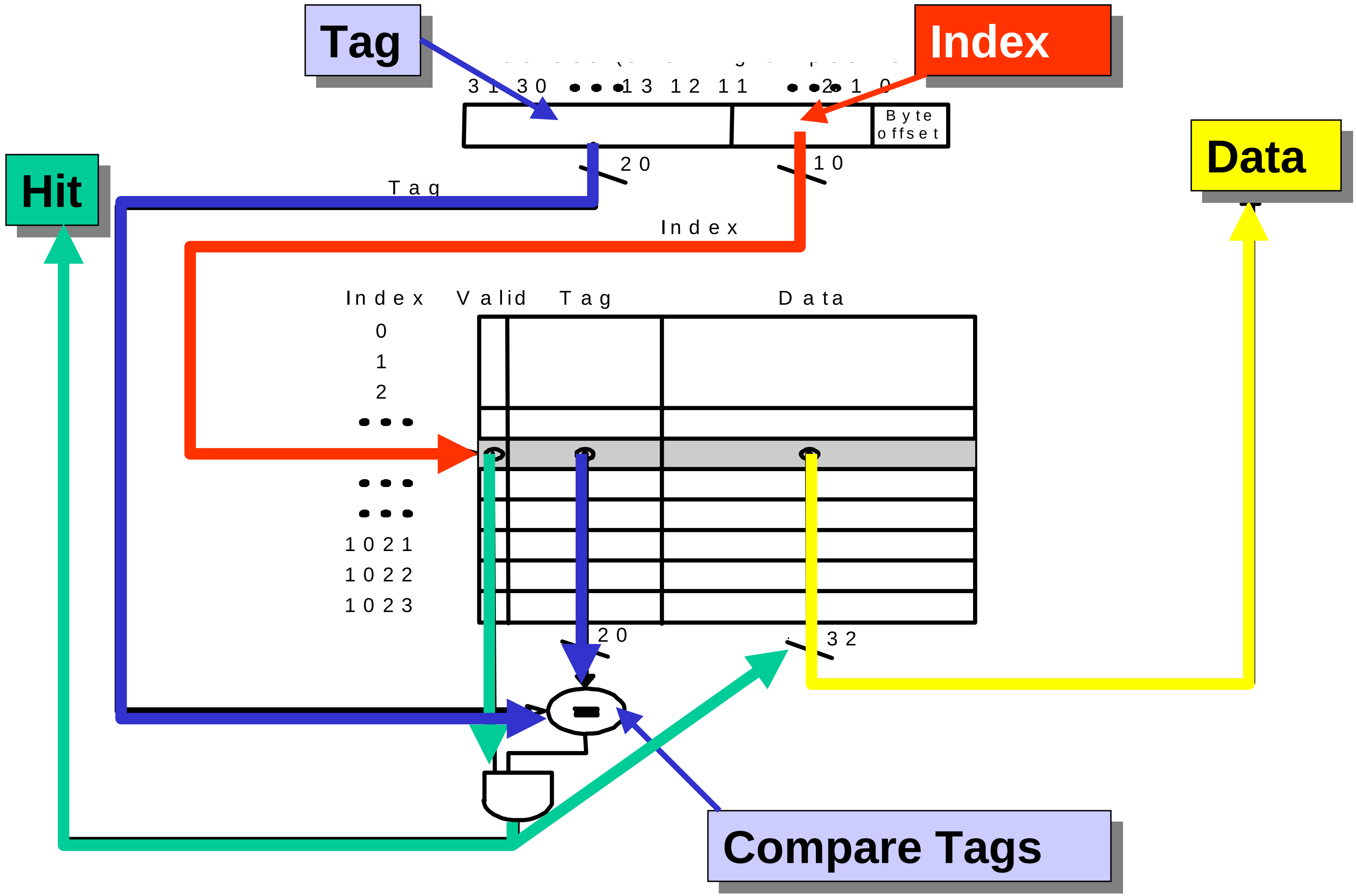
Figure 7.6

lw	$1,10 110 ($0)	Miss: valid	lw $1,22($0)
lw	$2,11 110 ($0)	Miss: tag	lw $2,30($0)
lw	$3,00 110 ($0)	Miss: tag	lw $3,6($0)

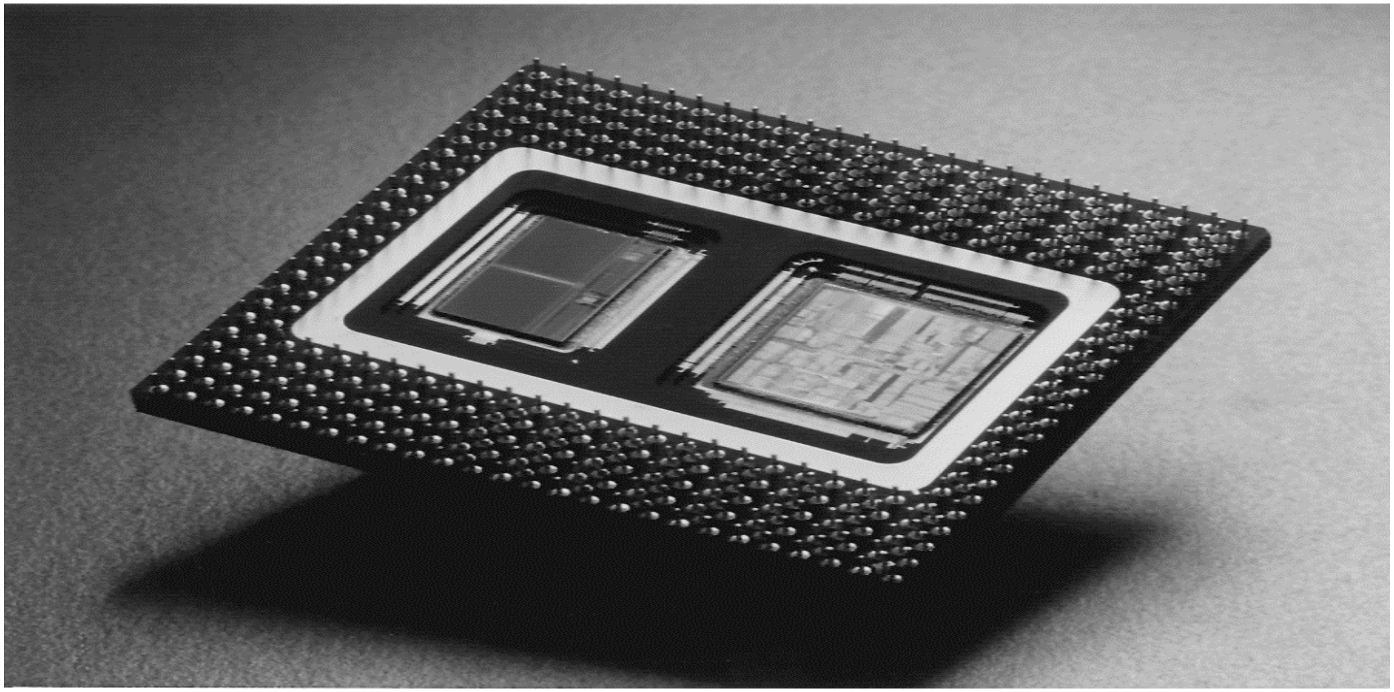
Index	Valid	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	00	Memory[00110]
111	N		

Direct Mapped Cache: Mips Architecture

Figure 7.7



Modern Systems: Pentium Pro and PowerPC



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through