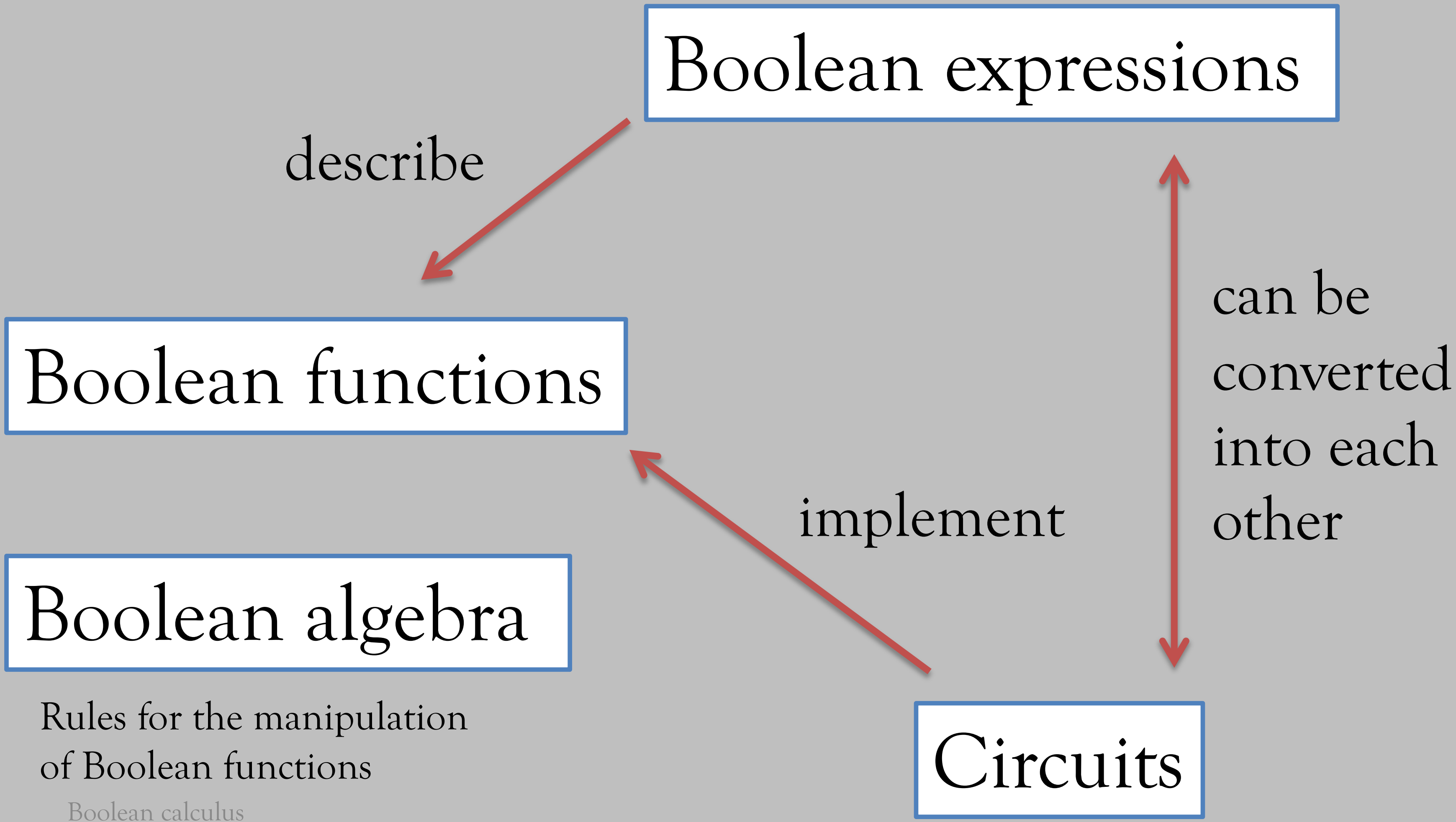
System Architecture
Summer Semester 2023 – Recap

Jan Reineke
Universität des Saarlandes

Reminder - Exam Registration

- Final Exam: July 25, 2023, 10:00-12:00
- Registration in LSF: July 19, 2023 (Today!)

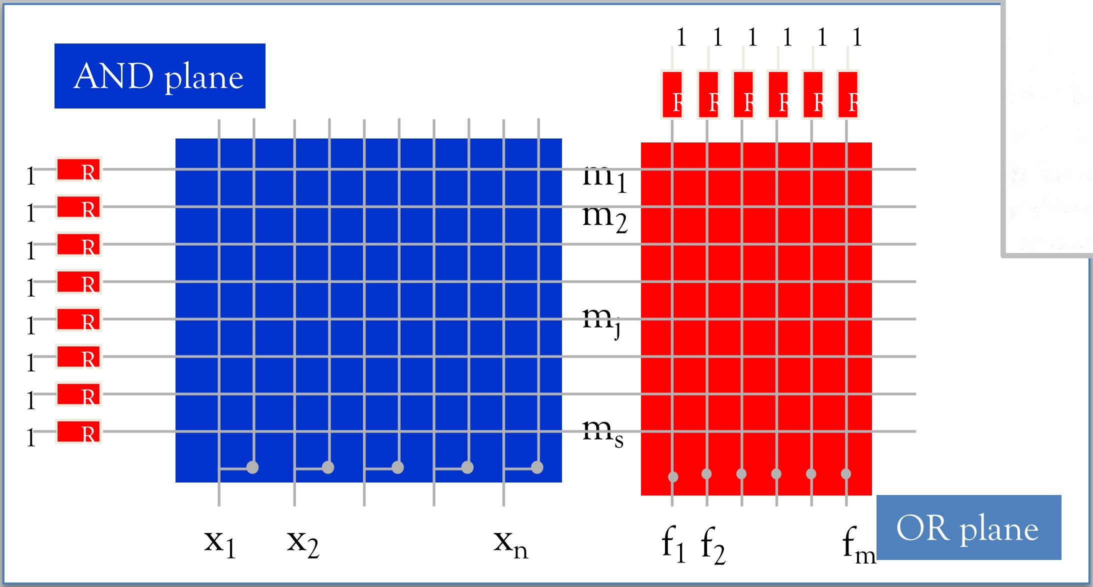
1. Boolean Calculus

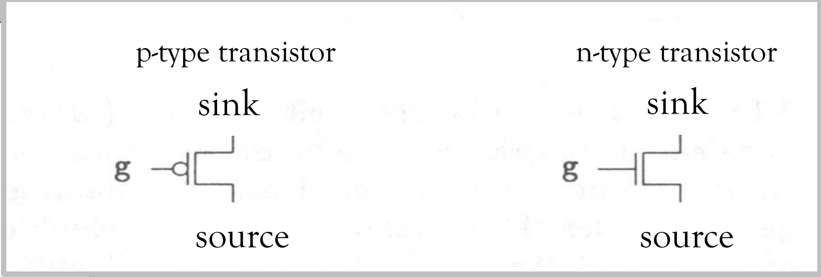

Boolean expressions
describe
Boolean functions
implement
can be converted into each other
Boolean algebra
Rules for the manipulation of Boolean functions
Circuits

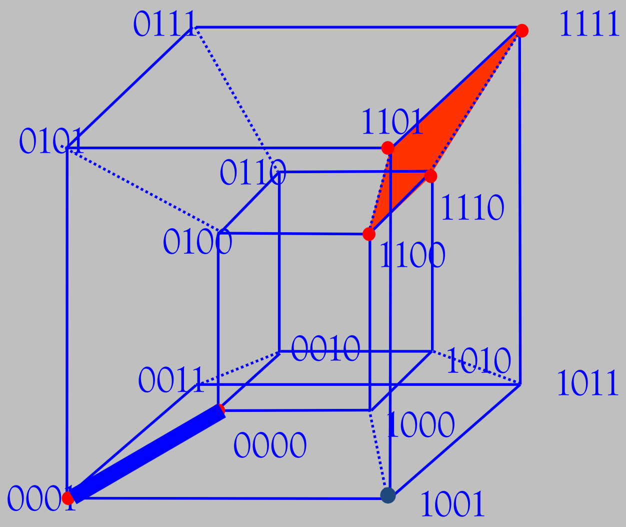
1. Boolean Calculus – Key Items

- Partial and total Boolean functions
- Truth tables, On-Set, Off-Set
- Boolean algebra, axioms, laws, duality principle
- Boolean expressions, syntax, semantics
- Literals, monomials, polynomials
- (Canonical) disjunctive/conjunctive normal form

2. PLAs and Logic Synthesis







Example:

$f(x_1,x_2,x_3,x_4)$

$= \boxed{x_1x_2}$

$\boxed{+\ x_1'x_2'x_3'}$

$\boxed{+\ x_1x_2'x_3'x_4}$

2. PLAs and Logic Synthesis – Key Items

- n-type and p-type transistors
- Programmable Logic Arrays (PLAs)
- Implementation of monomials and polynomials in PLAs
- Cost of monomials and polynomials
- Two-level logic minimization, and the corresponding covering problem on the hypercube

3. Implicants and Prime Implicants

An **implicant of f** is a monomial q with $\psi(q) \leq f$.
A **prime implicant of f** is a maximal implicant q of f.

Theorem (Quine):
Every minimal polynomial p of a Boolean function f consists only of prime implicants of f.

Theorem (Implicants):

A monomial m is an implicant of f if and only if, either

- m is a minterm of f, or
- $m \cdot x$ and $m \cdot x'$ are implicants of f
  for a variable x that does not occur in m.

Thus: $m \in \text{Implicant}(f) \Leftrightarrow$
$[m \in \text{Minterm}(f)] \vee [m \cdot x,\ m \cdot x' \in \text{Implicant}(f)]$

3. Implicants and Prime Implicants – Key Items

- Implicants and prime implicants
- Minimal polynomial
- Theorem of Quine
- Characterization of implicants

4. Quine/McCluskey Algorithm

```
Quine-Prime-Implicants($f$: $\mathbf{B}^n$ -> $\mathbf{B}$)
    $L_0$ := Minterm($f$)
    $i := 1$
    $Prime(f) := \emptyset$
    while ($L_{i-1} \neq \emptyset$) and ($i \leq n$)
        $L_i := \{m \mid |m| = n-i,\ m \cdot x \text{ and } m \cdot x' \text{ are in } L_{i-1} \text{ for some } x\}$
        $P_i := \{m \mid m \in L_{i-1} \text{ and } m \text{ is not covered by any } m' \in L_i\}$
        $Prime(f) := Prime(f) \cup P_i$
        $i := i+1$
    return $Prime(f) \cup L_{i-1}$
```

Quine's algorithm

McCluskey's Improvement

Compare only those monomials

- that contain the **same variables**, and
- whose number of **positive literals differs by one.**

4. Quine/McCluskey Algorithm

Matrix-covering problem

1. Reduction Rule:
Remove from the prime implicant table **PIT(f)** all essential prime implicants and all minterms that are covered by these prime implicants.

2. Reduction Rule:
Remove all minterms from the prime implicant table **PIT(f)** that dominate another minterm in **PIT(f)**.

3. Reduction Rule
Remove all prime implicants from the prime implicant table PIT(f) that are dominated by other prime implicants that are not more expensive.

4. Quine/McCluskey Algorithm – Key Items

- Quine’s algorithm
- McCluskey’s improvement
- Correctness and complexity of the Quine-McCluskey algorithm
- The matrix covering problem
- Three reduction rules
  - Essential prime implicants
  - Column domination
  - Row domination
- Cyclic covering problems, Petrick’s method

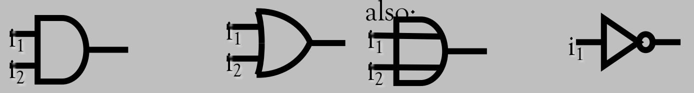
5. Combinatorial Circuits

Logic gates

i_2	i_1	AND_2
0	0	0
0	1	0
1	0	0
1	1	1

i_2	i_1	OR_2
0	0	0
0	1	1
1	0	1
1	1	1

i	NOT
0	1
1	0



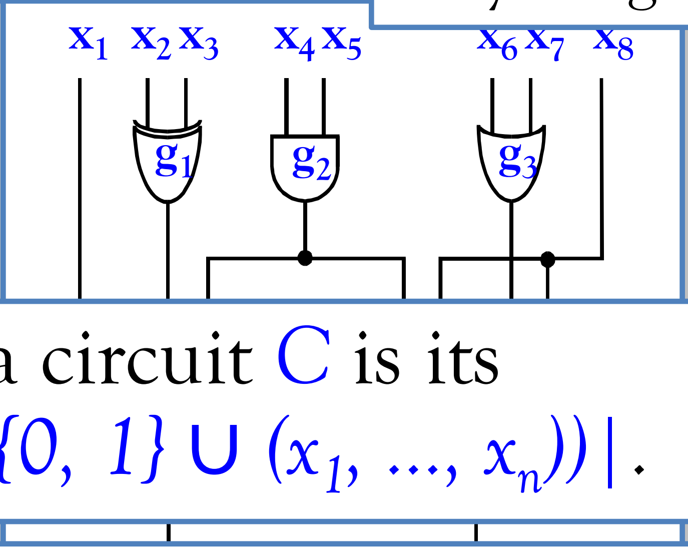
i_2	i_1	$NAND_2$
0	0	1
0	1	1
1	0	1

i_2	i_1	NOR_2
0	0	1
0	1	0
1	0	0

i_2	i_1	XOR_2
0	0	0
0	1	1
1	0	1

Circuits and their Formalization

“acyclic graph”



The **hardware cost** C(C) of a circuit C is its number of gates $|I| = |V \setminus (\{0, 1\} \cup (x_1, \ldots, x_n))|$.

The **depth** depth(C) of a circuit C is the **maximal number of gates on a path** from an arbitrary input x_i to an arbitrary output y_j of C.

5. Combinatorial Circuits – Key Items

- Logic gates, cell library
- Circuits
- Semantics of circuits
- Concrete and symbolic simulation
- Cost and depth of circuits
- Hierarchical circuits
- Circuits vs Boolean functions
- Implementation or associative operations

6. Number Representations

Questions:

1. How to represent *natural numbers*?
2. How to represent *integers*?
   Challenge: negative numbers
3. How to represent *rational numbers*?
4. How to represent *very large* and very *small numbers*?

fixed-point numbers

floating-point numbers

Binary numbers: $\langle d_n d_{n-1} \ldots d_0 \rangle := \Sigma_{i=0,\ldots,n}\, d_i \cdot 2^i$

Two's complement: $[d_n d_{n-1} \ldots d_0]_2 := \Sigma_{i=0,\ldots,n-1}\, d_i \cdot 2^i - d_n \cdot 2^n$

Fixed-point numbers: $[d_n d_{n-1} \ldots d_0, d_{-1} \ldots d_{-k}]_2 := \Sigma_{i=-k,\ldots,n-1}\, d_i \cdot 2^i - d_n \cdot 2^n$

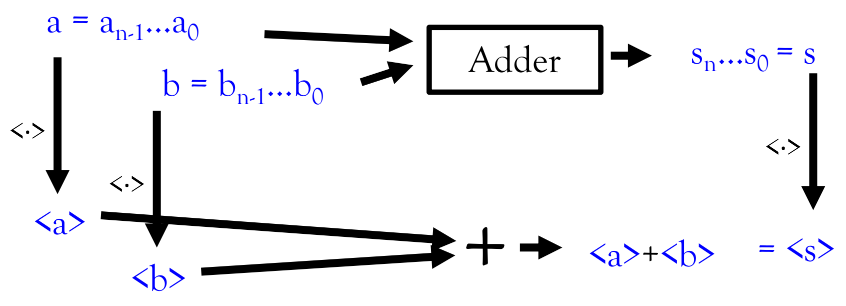
Floating-point numbers: sign, exponent, mantissa

6. Number Representations – Key Items

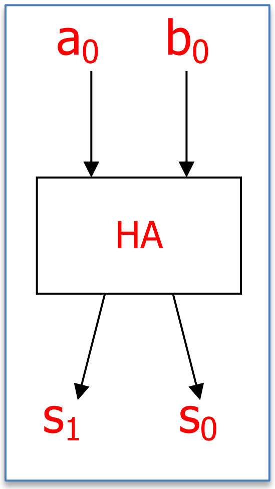
- Numerals (digits), binary, decimal, hexadecimal
- Positional numeral system
- Natural numbers
- Signed-magnitude representation, One's complement, Two's complement
- Fixed-point numbers
- Floating-point numbers

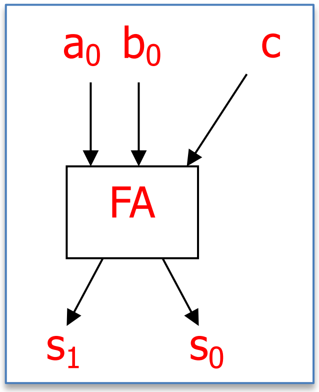
7. Arithmetic Circuits: Adders

Definition of an Adder

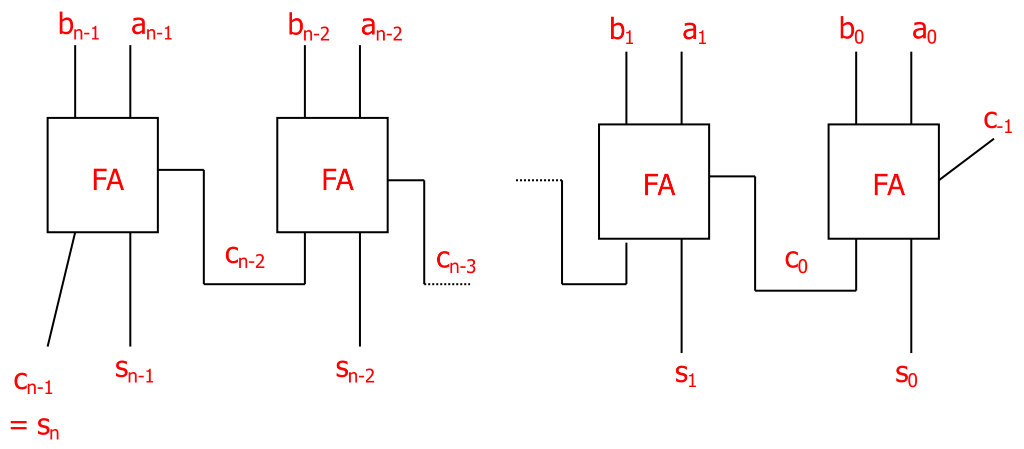


Half and Full adders





Ripple-carry Adders



$$C(RC_n) = n \cdot C(FA) = 5n$$

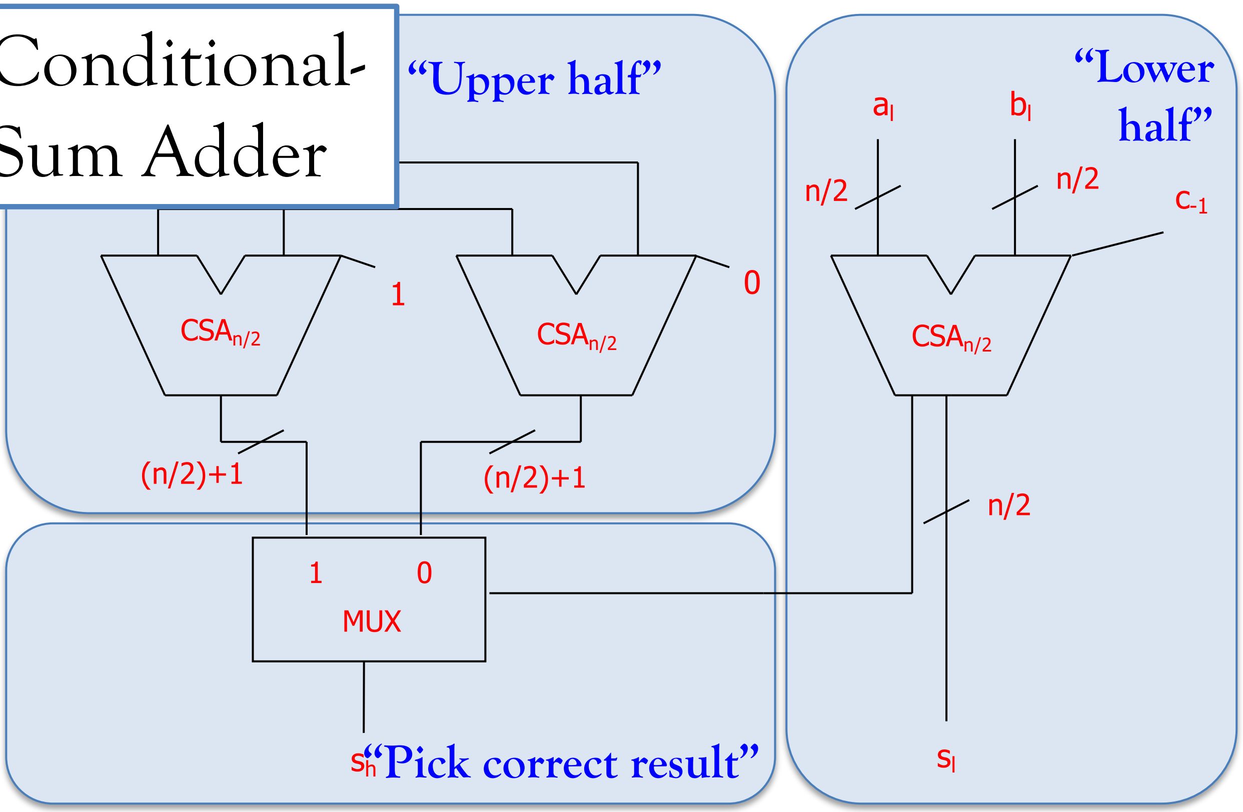
$$depth(RC_n) = 3 + 2(n-1)$$

7. Arithmetic Circuits: Adders

Lower bounds for adders!

$C(+_n) \geq 2n, \quad \text{depth}(+_n) \geq \log(n) + 1$

Conditional-Sum Adder



$\text{depth}(CSA_n) = 3 \log_2 n + 3$

$C(CSA_n) = 10n^{\log 3} - 3n - 2$

7. Arithmetic Circuits: Adders

Let M be a set and $\mathbf{o} : M \times M \rightarrow M$ an **associative** operation.
The **parallel prefix sum** $PP^n: M^n \rightarrow M^n$ is defined as follows:

$$PP^n(x_{n-1}, \ldots, x_0) = (x_{n-1} \mathbf{o}\, x_{n-2} \ldots \mathbf{o}\, x_0, \ldots, x_1 \mathbf{o}\, x_0, x_0)$$

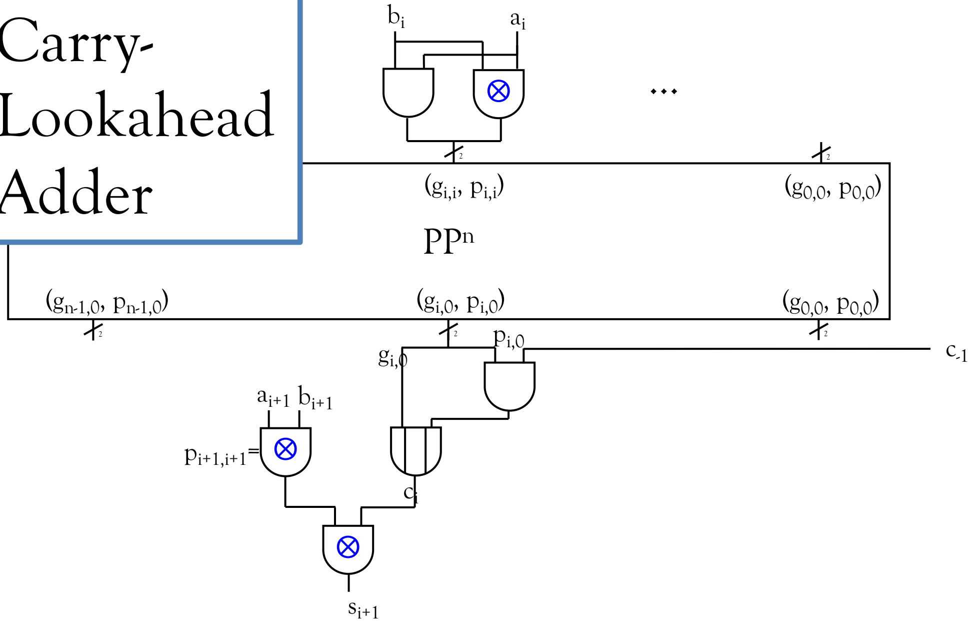
$\mathrm{depth}(PP^n) < (2 \cdot \log_2 n) \cdot \mathrm{depth}(\mathbf{o})$

$C(PP^n) < 2n \cdot C(\mathbf{o})$

Generated carry $g_{j,i}$ from i to j:
$c_j = 1$ *independently* of c_{i-1}.
Propagated carry $p_{j,i}$ from i to j:
$c_j = 1$ *if and only if* also $c_{i-1} = 1$

Generated and propagated carries can be captured as parallel prefixes of associative operator.

Carry-Lookahead Adder



7. Arithmetic Circuits: Adders – Key Items

- Definition of adder
- Half adder, Full adder
- Ripple-carry adder, correctness, cost, depth
- Recursive constructions, inductive proofs
- Incrementer, Multiplexer
- Lower bounds on cost and depth
- Conditional-sum adder, divide-and-conquer
- Addition in two's complement

7. Arithmetic Circuits: Adders – Key Items

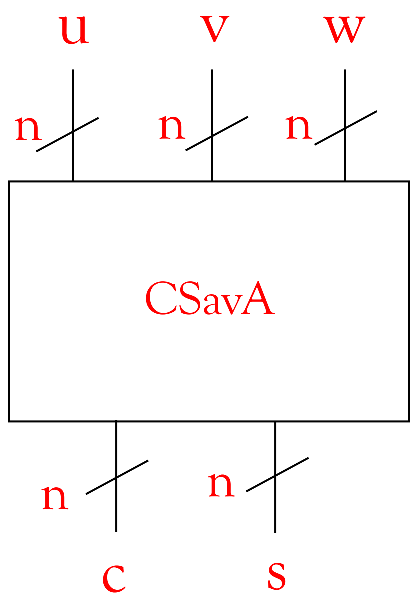
- Parallel prefix operation and circuit
- Generated and propagated carries
- Carry-lookahead adder

8. Arithmetic Circuits

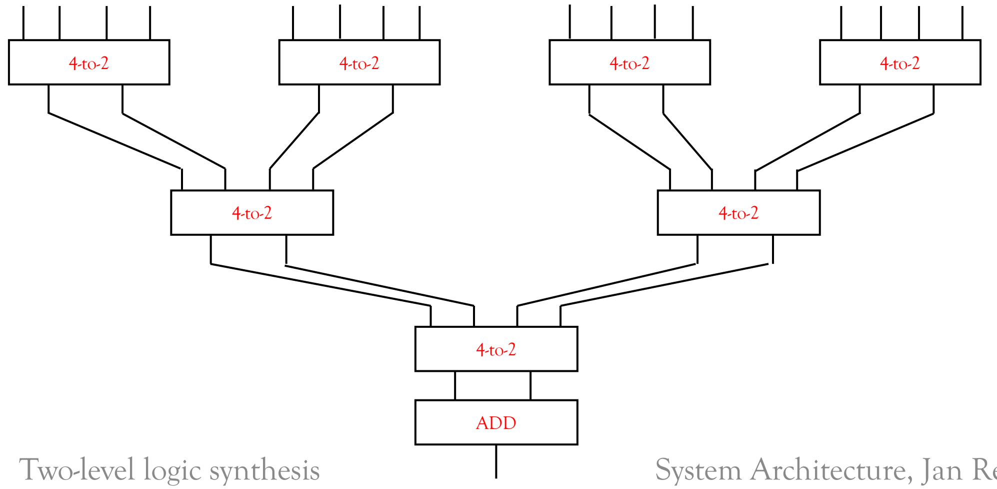
Multiplication matrix

$$\begin{pmatrix} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 & a_{n-1}b_0 & a_{n-2}b_0 & \dots & a_1b_0 & a_0b_0 \\ 0 & 0 & \dots & 0 & a_{n-1}b_1 & a_{n-2}b_1 & a_{n-3}b_1 & \dots & a_0b_1 & 0 \\ \vdots & & & & \vdots & \vdots & \vdots & & & \vdots \\ 0 & a_{n-1}b_{n-1} & \dots & a_2b_{n-1} & a_1b_{n-1} & a_0b_{n-1} & 0 & \dots & 0 & 0 \end{pmatrix}$$

Carry-save adder

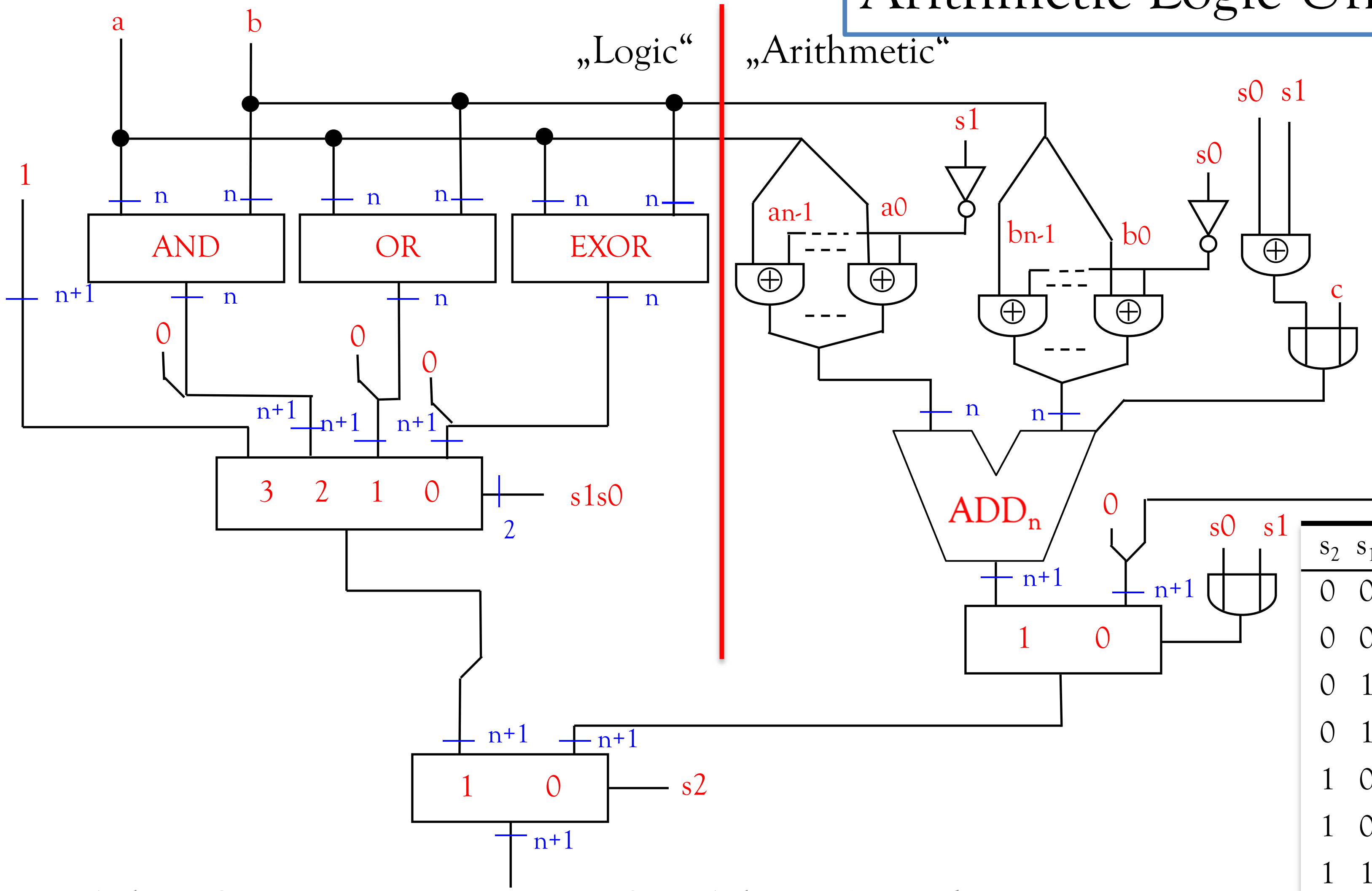


Adder stage of log-time multiplier



8. Arithmetic Circuits

Arithmetic Logic Unit (ALU)



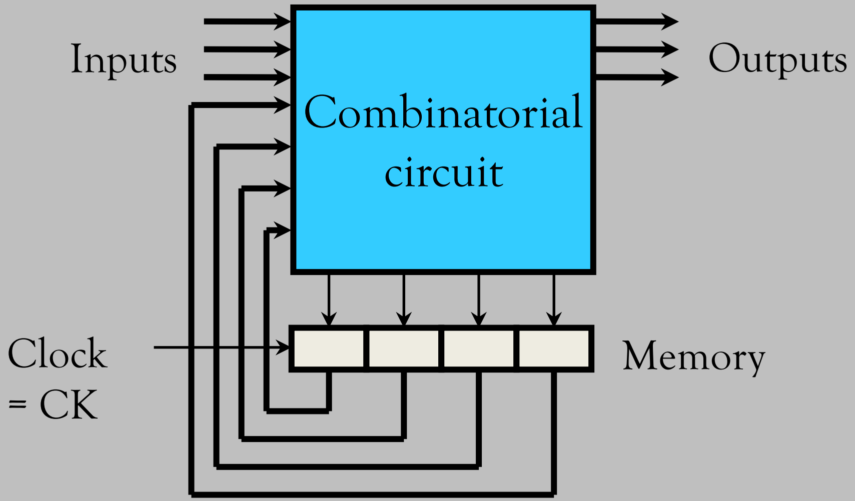
s_2	s_1	s_0	Function
0	0	0	0 … 0
0	0	1	[b] – [a]
0	1	0	[a] – [b]
0	1	1	[a] + [b] + c
1	0	0	a ⊕ b
1	0	1	a ∨ b
1	1	0	a ∧ b
1	1	1	1 … 1

8. Arithmetic Circuits – Key Items

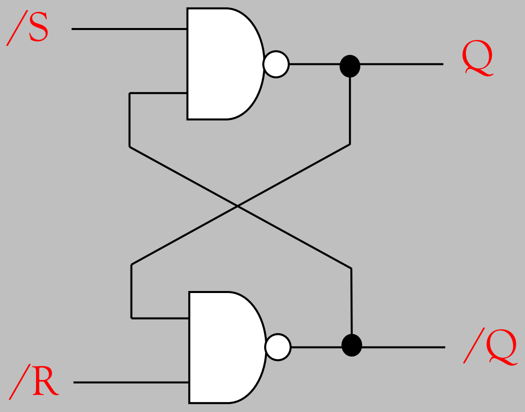
- Subtractor, combined adder/subtractor
- Multiplier
- Carry-save adder
- Arithmetic logic unit

9. Sequential Circuits

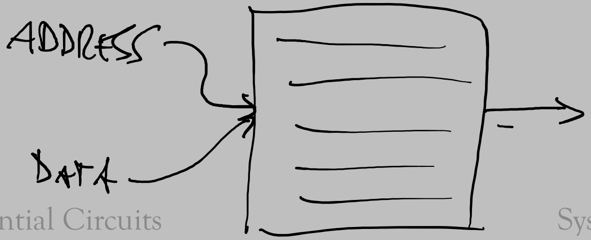
Sequential Circuit



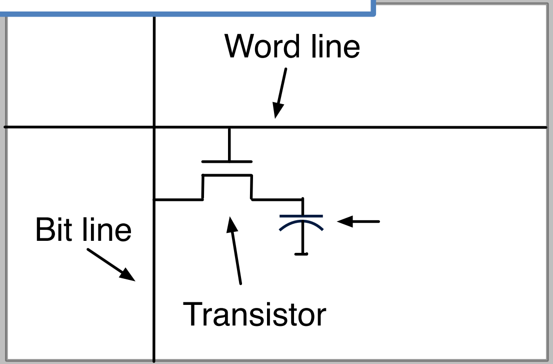
Latches and Flip-Flops



Random Access Memory



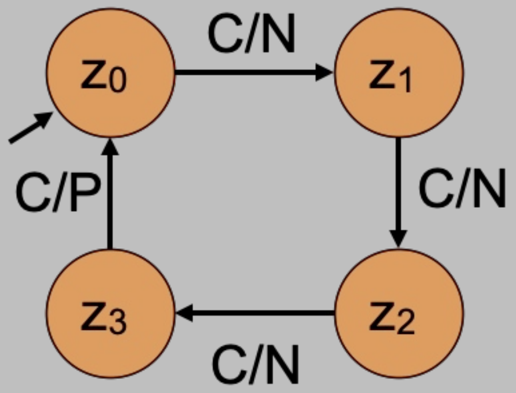
Dynamic RAM



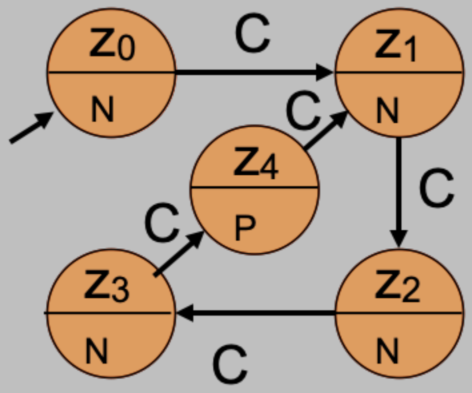
9. Sequential Circuits

$$\begin{tikzcd}
\textbf{Sequential circuits} \arrow[r, bend left, "\text{model/abstraction}"] & \textbf{Finite state machines} \arrow[l, bend left, "\text{implementation/synthesis}"]
\end{tikzcd}$$

Mealy Machines



Moore Machines

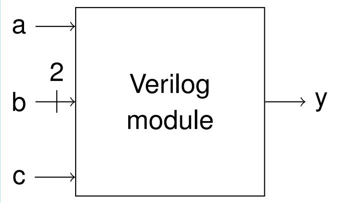


9. Sequential Circuits – Key Items

- SR latch, D latch, D flip-flop
- Register
- Random access memory (RAM)
- Decoder
- Static RAM and Dynamic RAM
- Sequential circuits
- Finite state machines, Mealy and Moore machines
- Synthesis of sequential circuits

10. Verilog

Encapsulation

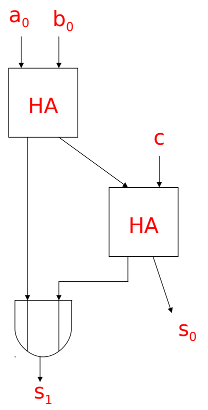


```
module name(
   input a,
   input [1:0] b,
   input c,
   output y
);

// functionality

endmodule
```

Hierarchical circuits



```
module fulladder(
   input a0,
   input b0,
   input c,
   output s0,
   output s1
);
   wire ha0c, ha0s, ha1c;
   halfadder ha0(.a(a0), .b(b0), .c(ha0c), .s(ha0s));
   halfadder ha1(.a(ha0s), .b(c), .c(ha1c), .s(s0));
   assign s1 = ha0c | ha1c;
endmodule
```

Blocking vs non-blocking assignments

Blocking assignment

```
reg x, y;
always @(posedge clk)
begin
   x = y;
   y = x;
end
```

Assignments are blocking, i.e., they are executed **sequentially**
⇒ *x* and *y* are equal

Non-blocking assignment

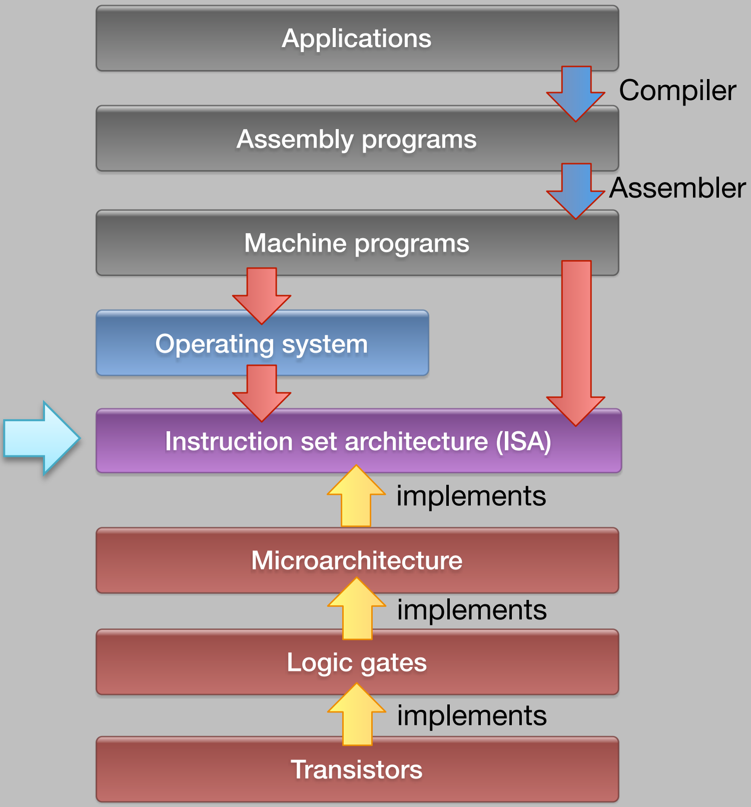
```
reg x, y;
always @(posedge clk)
begin
   x <= y;
   y <= x;
end
```

Assignments are non-blocking, i.e., are conceptually executed in parallel
⇒ *x* and *y* are exchanged

10. Verilog – Key Items

- Hardware description languages
- Simulation and hardware synthesis
- Blocking vs non-blocking assignments
- Test benches

11. Instruction Set Architecture



Instruction set architecture (or just "architecture")
= set of instructions, their **encoding** and **semantics**
= „**What**" a computer computes
For example: x86, ARM

Assembly language = textual representation

↓ Assembler + Linker

Machine language = binary representation

MIPS instruction set
- instructions
- encoding

11. Instruction Set Architecture – Key Items

- Instruction set architecture
- Assembly language
- Machine language, instruction encoding
- MIPS instruction set
- Addressing modes
- Little endian, big endian
- Sign extension

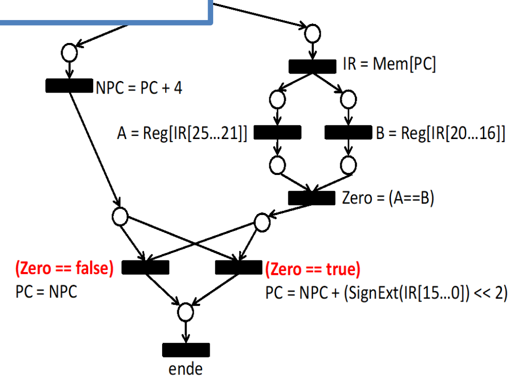
12. Microarchitecture

Microarchitecture

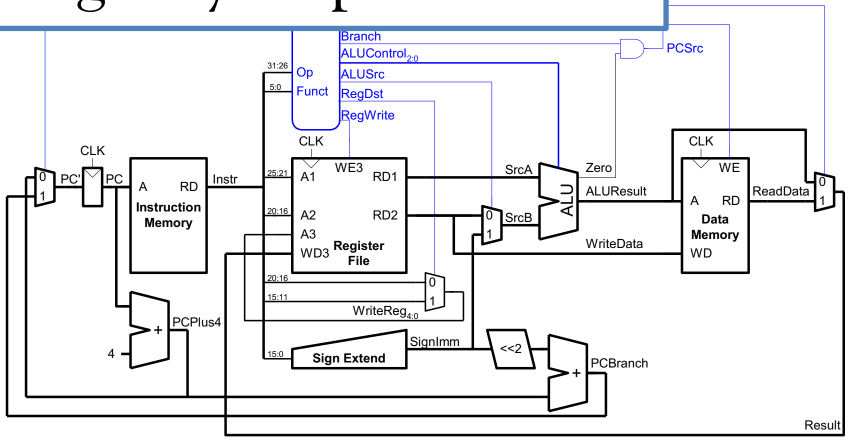
= concrete implementation of an instruction set in hardware

= „**How**" a computer works; e.g. Intel Skylake, AMD Zen 3, Apple M1

Petri nets



Single-cycle processor



12. Microarchitecture – Key Items

- Microarchitecture
- Datapath, control
- Petri nets
- Single-cycle system
- Main decoder, ALU decoder
- ALU implementation

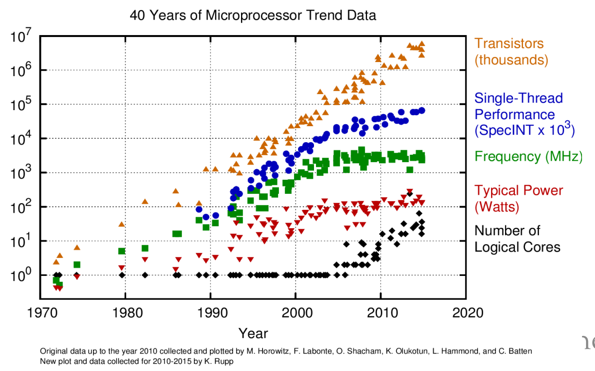
13. Performance: Basic Concepts

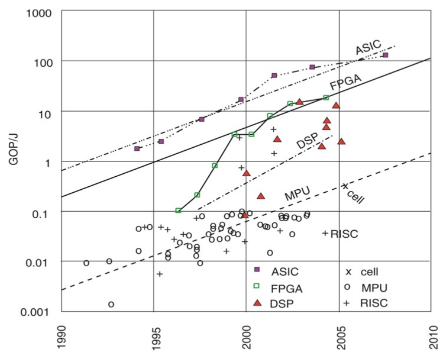
Latency = time required to perform a single task

Throughput = number of tasks performed in one time unit

Processor time = Number of executed instructions
* Cycles per instruction
* Cycle time

Technological developments

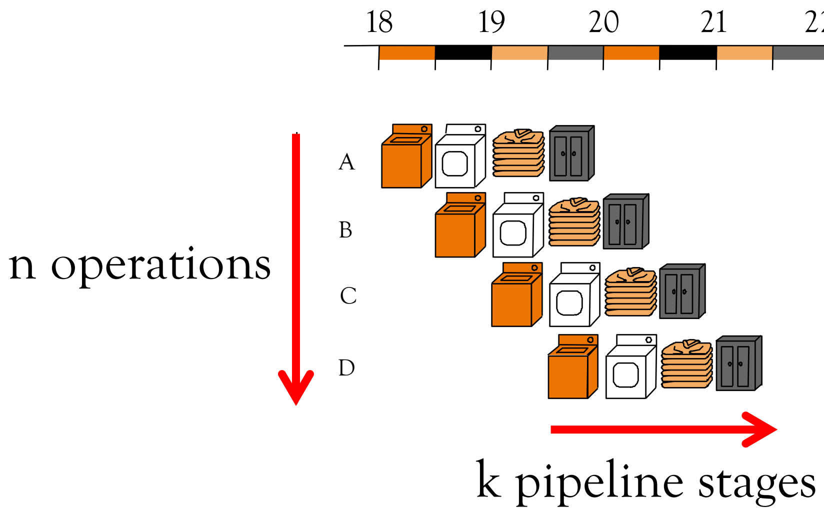




13. Performance: Basic Concepts – Key Items

- Performance definitions, latency, throughput
- Execution time, response time, processor time
- Cycles per instruction, cycle time
- Moore's law
- Reduced Instruction Set Computer (RISC), Complex Instruction Set Computer (CISC)
- Pipelining

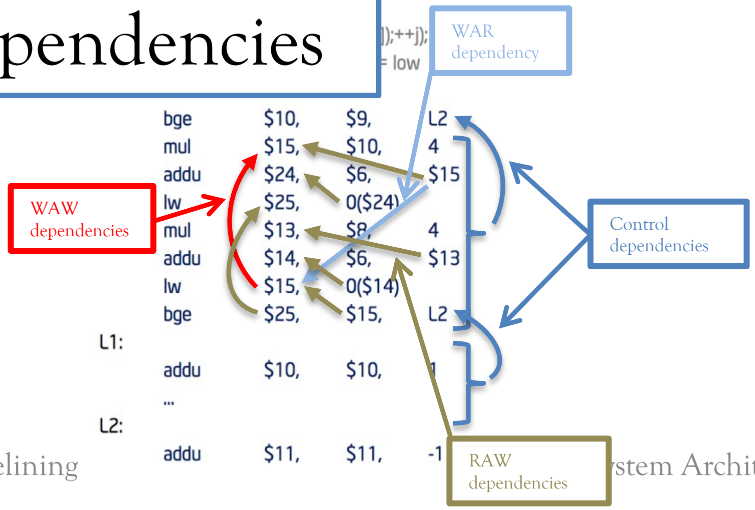
14. Pipelining



$$Speedup = \frac{n \cdot k}{k+n-1} \xrightarrow{n \to \infty} k$$

$$\text{Efficiency} = \frac{n \cdot k}{(k+n-1) \cdot k} = \frac{Speedup}{k} \xrightarrow{n \to \infty} 1$$

Dependencies



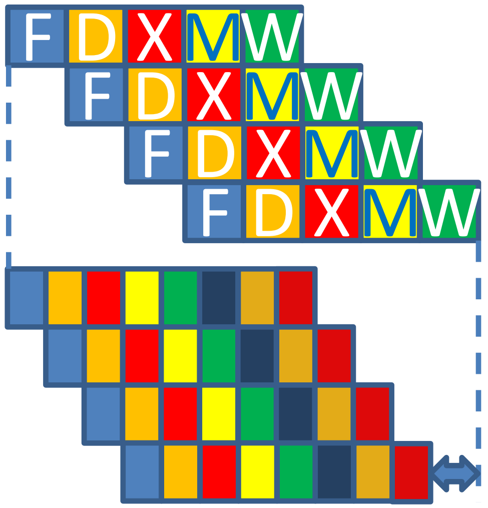
```
      bge   $10, $9,  L2
      mul   $15, $10, 4
      addu  $24, $6,  $15
      lw    $25, 0($24)
      mul   $13, $8,  4
      addu  $14, $6,  $13
      lw    $15, 0($14)
      bge   $25, $15, L2
L1:
      addu  $10, $10, 1
      ...
L2:
      addu  $11, $11, -1
```

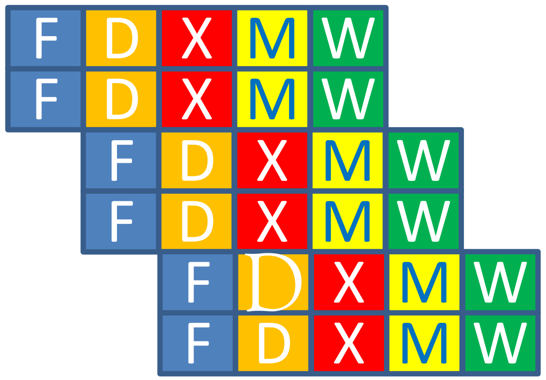
+ Hazards

Stalls
Forwarding
Branch prediction

15. Advanced Pipelining Concepts

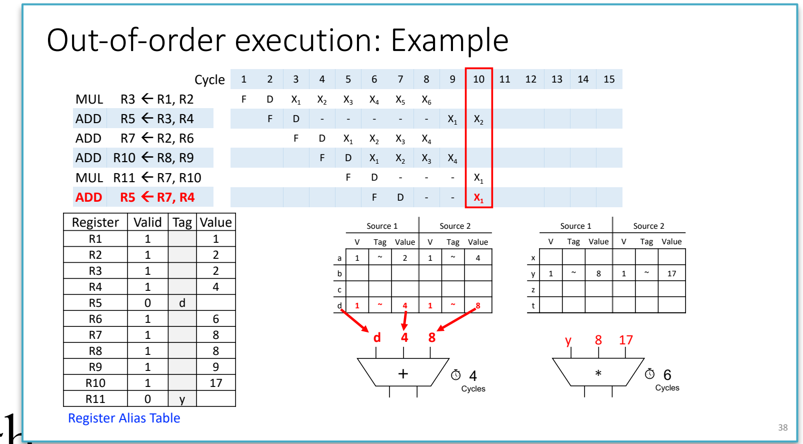
Deep + Superscalar Pipelines





Best case CPI = 1/width

Tomasulo Algorithm



Out-of-order execution: Example

	Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MUL	R3 ← R1, R2	F	D	X_1	X_2	X_3	X_4	X_5	X_6							
ADD	R5 ← R3, R4		F	D	-	-	-	-	-	X_1	X_2					
ADD	R7 ← R2, R6			F	D	X_1	X_2	X_3	X_4							
ADD	R10 ← R8, R9				F	D	X_1	X_2	X_3	X_4						
MUL	R11 ← R7, R10					F	D	-	-	-	X_1					
ADD	**R5 ← R7, R4**						F	D	-	-	X_1					

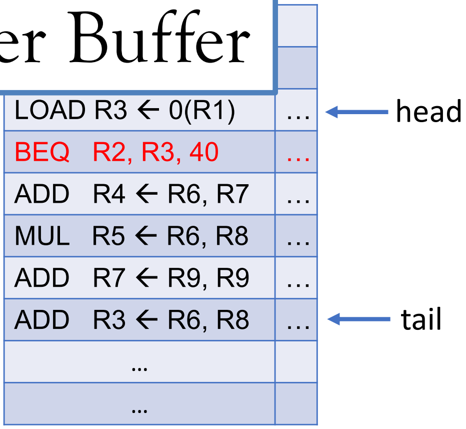
Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	0	y	

Register Alias Table

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b						
c						
d	1	~	4	1	~	8

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x						
y	1	~	8	1	~	17
z						
t						

Reorder Buffer



LOAD R3 ← 0(R1)	…	← head
BEQ R2, R3, 40	…	
ADD R4 ← R6, R7	…	
MUL R5 ← R6, R8	…	
ADD R7 ← R9, R9	…	
ADD R3 ← R6, R8	…	← tail
…		
…		

Speculative Execution

Predict outcome of branches

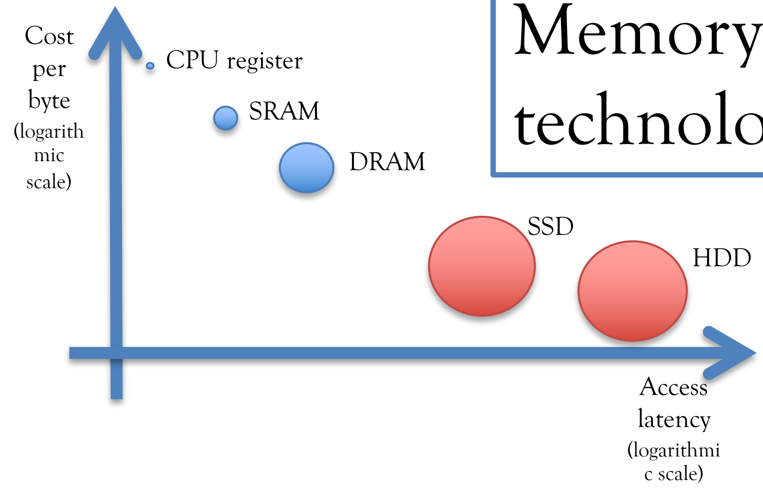
+ Execute instructions speculatively

15. Advanced Pipelining Concepts – Key Items

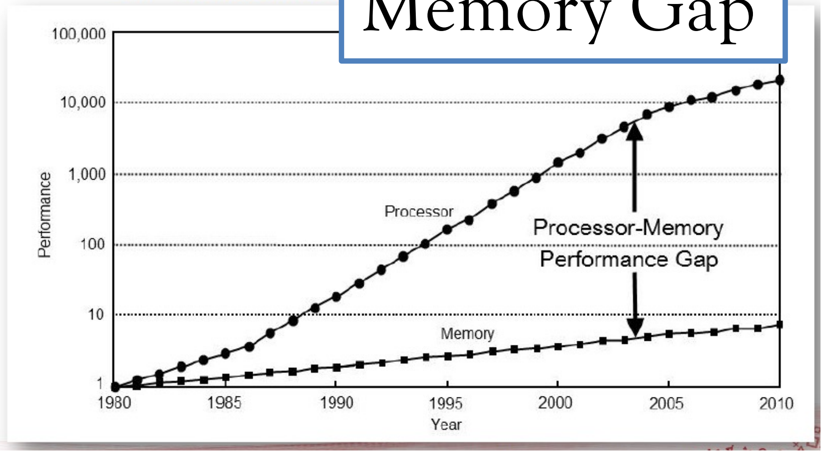
- Flynn bottleneck
- Out-of-order execution
- Reservation stations
- Tomasulo algorithm
- Speculative execution

16. Memory Hierarchy, Caches

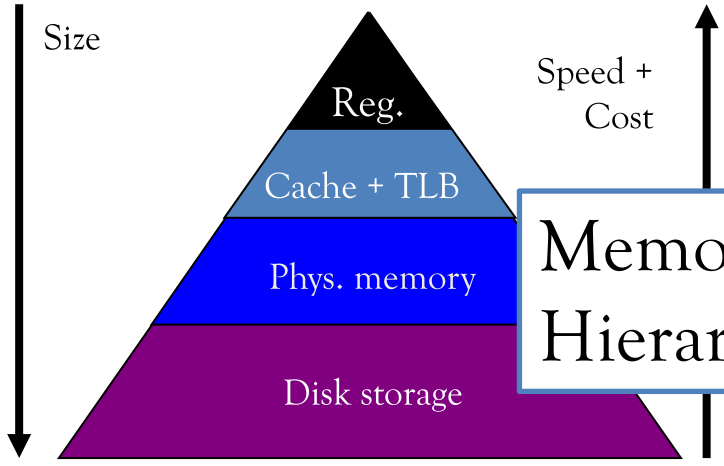
Memory technologies



Memory Gap



Memory Hierarchy

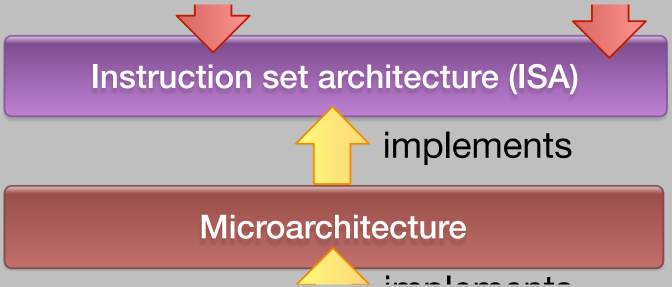


16. Memory Hierarchy, Caches – Key Items

- Memory technologies, Memory Gap
- Memory Hierarchy
- Scratchpad memory, Caches
- Fully-associative, direct-mapped, set-associative
- Replacement policy
- Optimal replacement, Farthest-in-the-Future
- Least-recently-used (LRU), First in, first out (FIFO)
- Online and offline algorithms
- Temporal and spatial locality

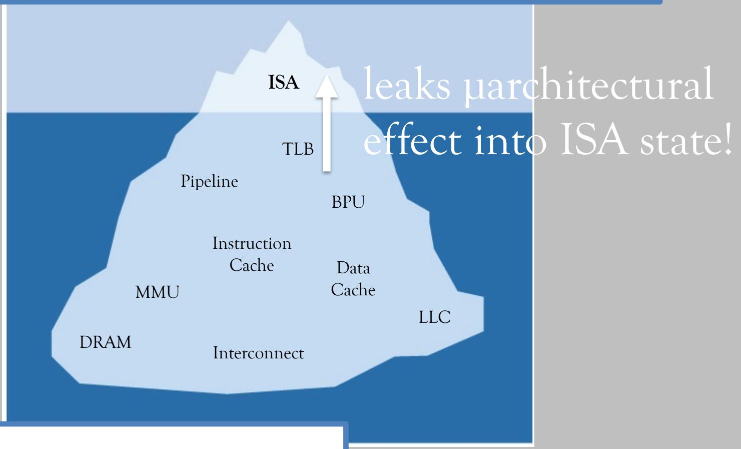
17. ISA, µArchitecture, and Bad News from the Real World

Correct ISA Implementation?



Leaky Abstractions

`rdtsc`: "read time-stamp counter"



Flush+Reload

1. FLUSH memory line
2. Wait a bit
3. Measure time to RELOAD line

Spectre Attack

„JavaScript"-Code:

```
if (offset < bound) {
    value = some_array[offset];
    tmp = other_data[(value>>bit)&1];
}
```

Extracts a bit of "value"

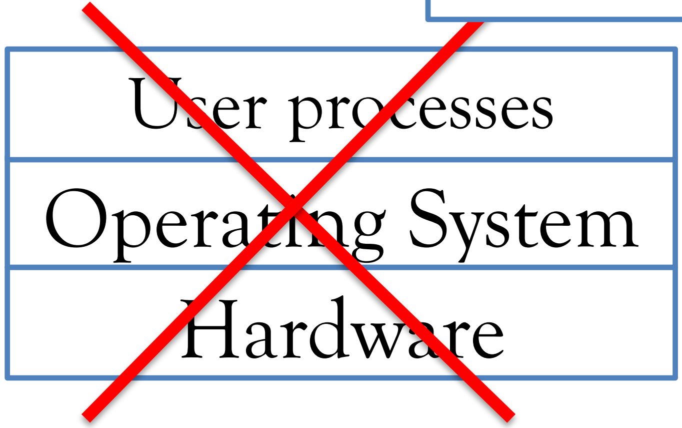
2. Secret-dependent memory access

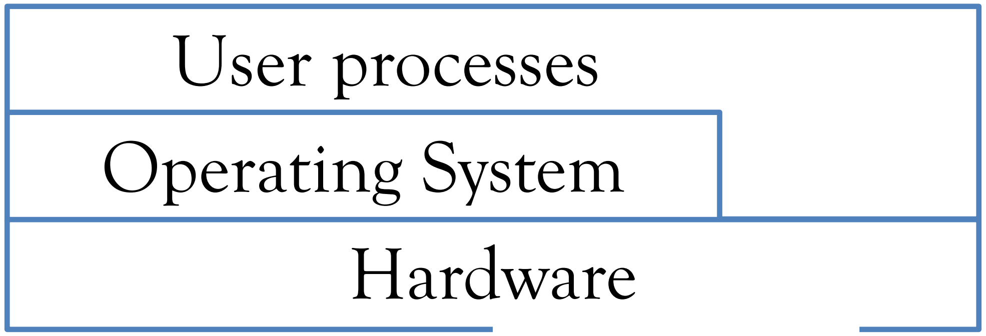
17. ISA, µArchitecture, and Bad News from the Real World – Key Items

- Correct ISA implementation
- Flush+Reload
- Prime+Probe
- Spectre attack

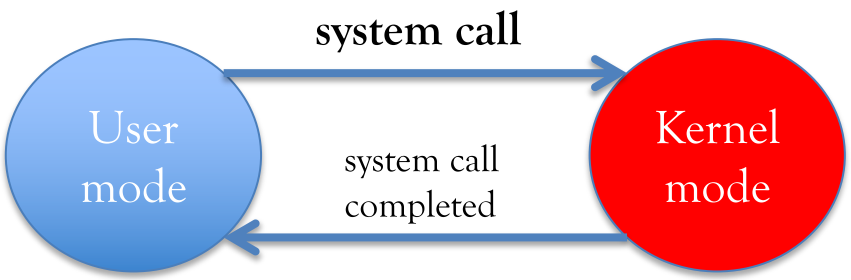
19. Virtualization: The CPU

Direct execution





Limited direct execution

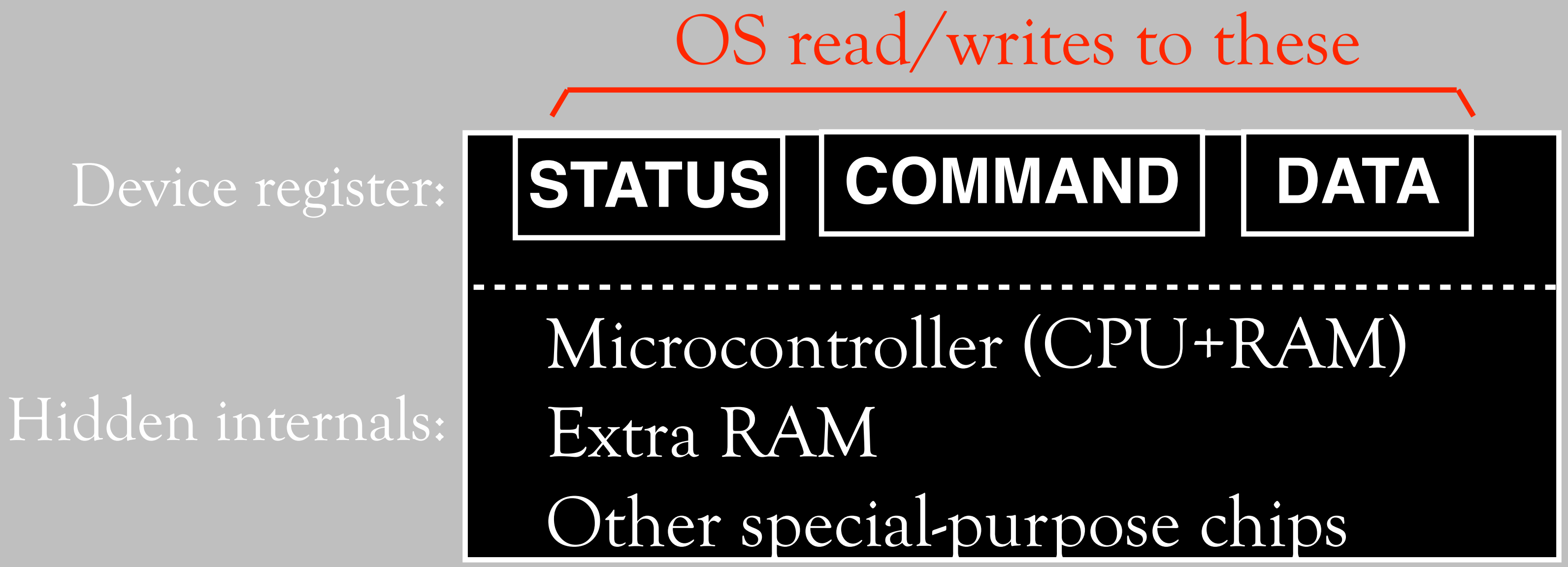


Preemptive Multitasking

19. Virtualization: The CPU – Key Items

- Process, process vs program
- Direct execution
- Restricted operations
- User mode vs kernel mode
- System calls, exception handling
- Mechanism vs policy
- Context switches
- Cooperative vs preemptive multitasking

20. Persistence: I/O Devices



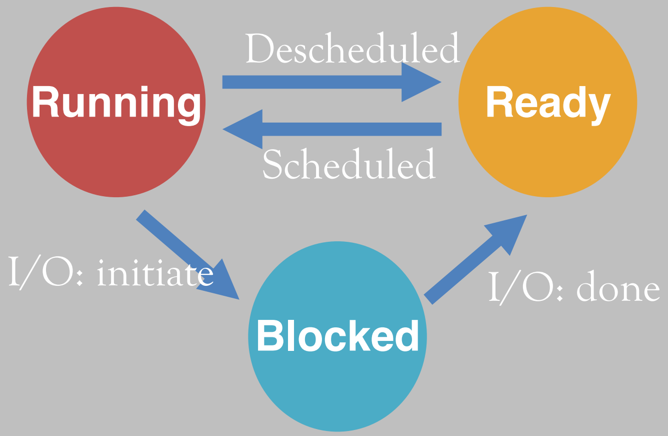
- **Status checks**: polling *vs.* interrupts
- **Data**: Programmed-IO *vs.* DMA
- **Control**: special instructions vs. memory-mapped I/O

20. Persistence: I/O Devices – Key Items

- I/O devices
- Busy waiting/polling vs interrupts
- Programmed I/O vs Direct Memory Access (DMA)
- Drivers

21. Scheduling

Process state

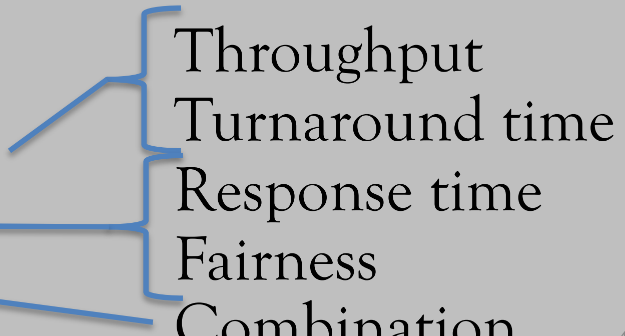


Performance metrics

Turnaround time
Response time
Throughput
Fairness
Meet deadlines

Scheduling policies

First Come, First Served (FCFS)
Shortest Job First (SJF)
Shortest Time-to-Completion First (STCF)
Round Robin
Multi-Level Feedback Queue (MLFQ)



21. Scheduling – Key Items

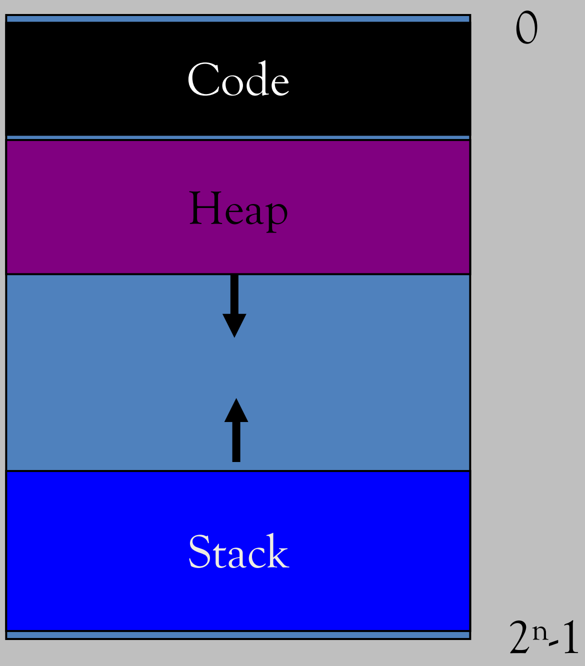
- Dispatcher vs Scheduler
- Workload, Performance metric
- Turnaround time, Response time, Throughput, Overhead, Fairness
- FIFO (also FCFS), Convoy effect
- Shortest Job First (SJF), Shortest Time-to-Completion First (STCF)
- Round Robin
- Multi-Level Feedback Queue (MLFQ)
- Starvation
- Voodoo constants

22. Memory Virtualization Foundations

Virtualization Goals

Transparency
Protection
Efficiency
Sharing

(Virtual) address space



Mechanisms for virtual memory:

1. Time sharing
2. Static relocation
3. Dynamic relocation
4. Segmentation

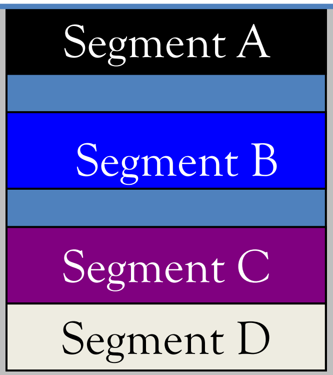
(Dis-)advantages of these?

22. Memory Virtualization Foundations – Key Items

- Transparency, protection, efficiency, sharing
- Address space
- Static: code and global data, dynamic: stack and heap
- Time sharing, Static relocation, Dynamic relocation, Segmentation
- Memory Management Unit (MMU)
- Base and bounds
- Segment table

23. Paging

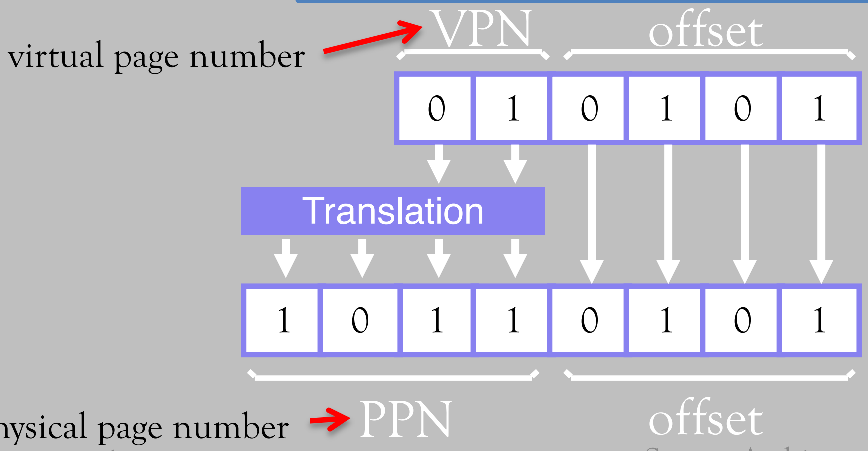
External and internal fragmentation

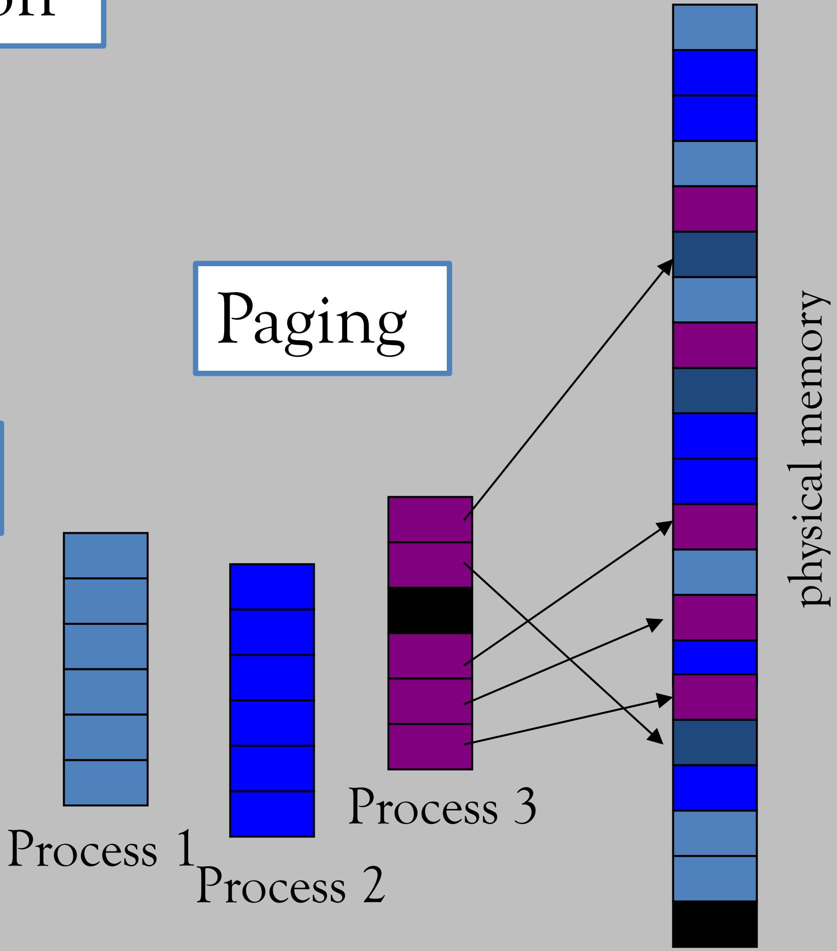


Allocated to application:

Paging

Address translation





23. Paging – Key Items

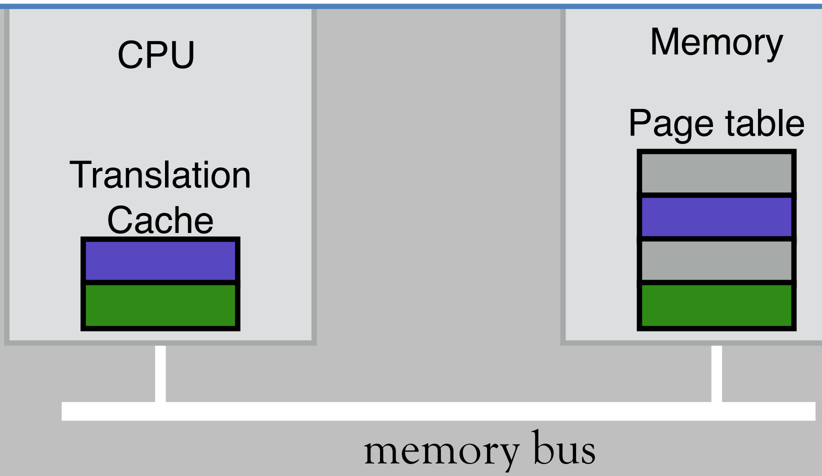
- Internal and external fragmentation
- Paging
- Pages, page frames
- Page number, frame number, page offset
- Virtual address, physical address
- Page table
- Valid bit, protection bits

24. Translation Lookaside Buffers

“Naïve” paging too slow

two physical accesses for every virtual access

Translation Lookaside Buffers



Design choices

page sizes
associativity
replacement policy

24. Translation Lookaside Buffers – Key Items

- Translation Lookaside Buffer
- Direct-mapped, fully-associative, set-associative
- Influence of page size on performance
- Influence of locality on performance
- TLB replacement policies
- Address space identifiers (ASIDs)
- TLB miss handling

25. Smaller Page Tables

Sizes of page tables

4 byte PTEs, **4 KB** pages

1. virt. addresses: **32 bits** — 4 MB
2. virt. addresses: **64 bits** — 2^{14} TB

VPN	valid	protection
10	1	r-x
-	0	-
23	1	rw-
-		
-		
-	0	
-	0	-
…many invalid entries…		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

Address spaces sparsely populated

Hierarchical page tables

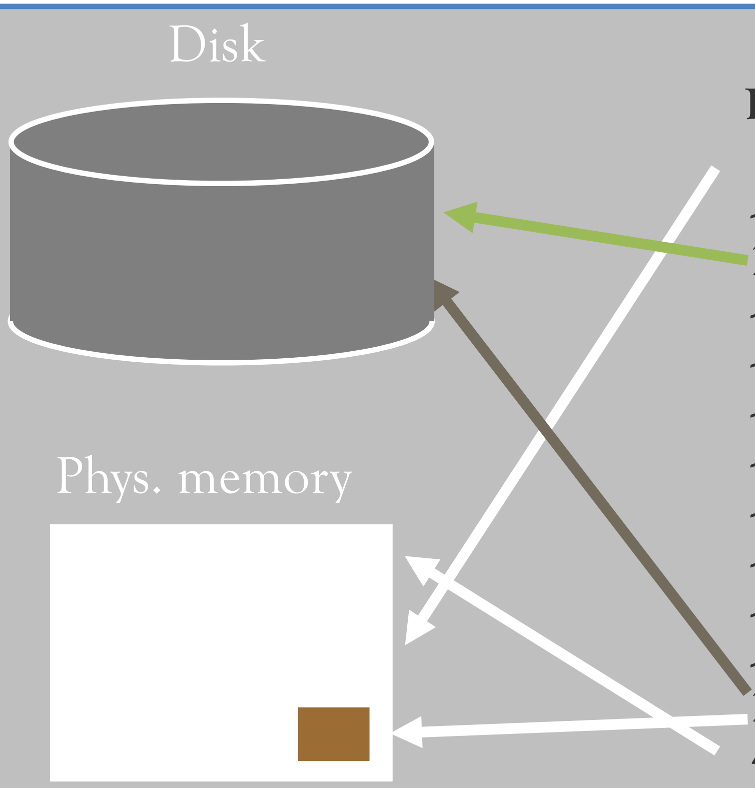
Segmented page tables
Multi-level page tables

25. Smaller Page Tables – Key Items

- Invalid page table entries
- Segmented page tables
- Multi-level page tables
- Outer page, inner page
- Page tables fit within pages

26. Swapping

Pages can be in memory or on disk



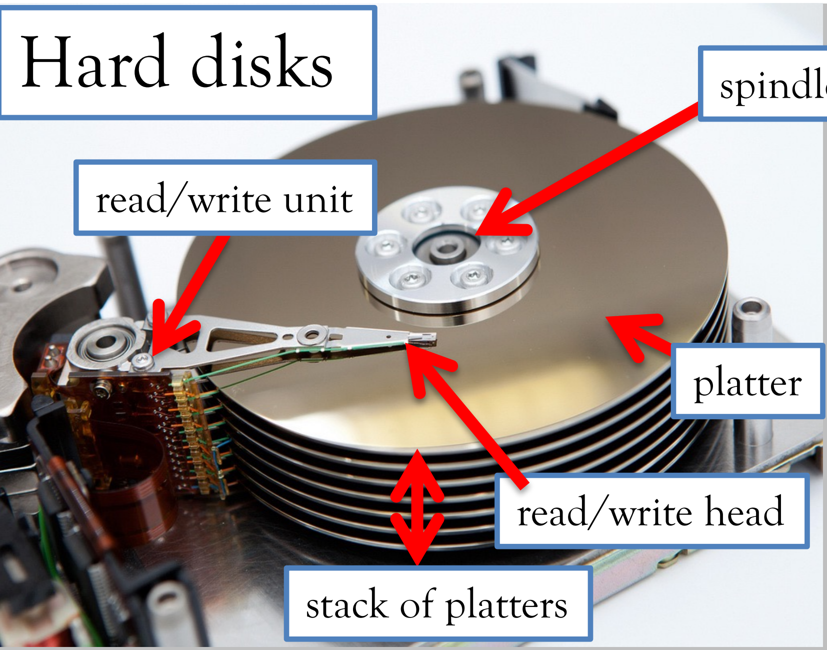
PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
16 28	1	rw-	1 0
4	1	rw-	1

26. Swapping– Key Items

- Swapping
- Present bit in page table
- Page fault
- HW + OS cooperate on address translation
- Precise interrupts
- Page selection and Page replacement
- Demand paging, Prefetching, Hints
- Clock algorithm

28. Persistence: Disks + I/O Scheduling

Hard disks



Time to read/write

Seek → slow
Rotation → slow
Transfer time → fast

Performance depends on workload

Workload	Toshiba	Seagate Exos
Sequential	290 MB/s	261 MB/s
Random	1 MB/s	0,47 MB/s

I/O Scheduling

Shortest Positioning Time First
SCAN algorithms
Anticipatory schedulers

28. Persistence: Disks + I/O Scheduling – Key Items

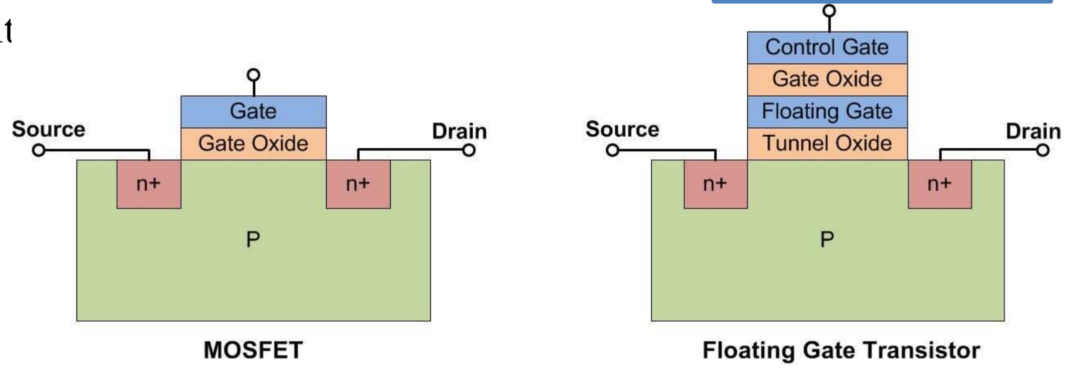
- Persistent vs volatile memory
- platter, surface, spindle, cylinder, track, sector, read/write unit, read/write head
- Seek, rotation, and transfer
- Throughput on sequential and random workloads
- Shortest Positioning Time First (SPTF), Shortest Seek Time First (SSTF)
- SCAN algorithms, Elevator algorithm, C-SCAN
- Work conservation, anticipatory schedulers

29. Persistence: Flash-based Solid State Disks

Solid-state storage devices

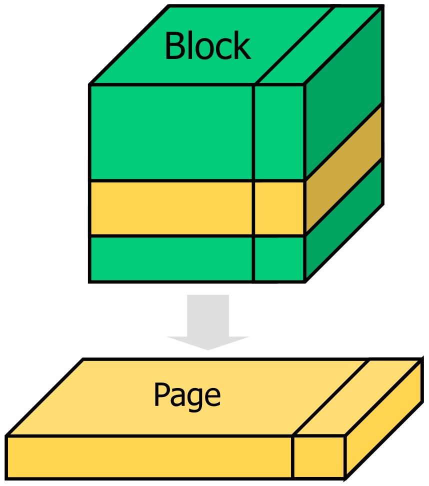
- No mechanical or moving parts like HDD
- Built out of transistors; but persistent unlike typical RAM

Flash cell



- electrons can be **trapped in** the floating gate
- electrons do not escape → **persistent memory**

Hierarchical organization



Read: at page granularity
Write: 1 → 0: at page granularity
Erase: 0 → 1: only at block granularity

29. Persistence: Flash-based Solid State Disks – Key Items

- Solid-state storage devices
- Floating-gate transistors
- Single-level cells, multi-level cells, etc.
- Basic operations: read, write, erase
- Reliability: wear out
- Out-of-place update
- Flash Translation Layer (FTL)

30. Error Detection and Correction

Hamming distance

dist(00001101,10001100) = *2*

Lemma (Error Detection)

A fixed-length code c is *k*-error detecting *iff* dist(c) ≥ k+1.

Parity code

Hamming code

Lemma (Error Correction)

A fixed-length code *c* is *k*-error correcting *iff* dist(c) ≥ 2k+1.

30. Error Detection and Correction – Key Items

- (Fixed-length) codes
- Hamming distance, code distance
- k-error detecting, k-error correcting
- Repetition code
- Parity code
- Hamming code

The End.