[Number System] 2's Comp. conversion

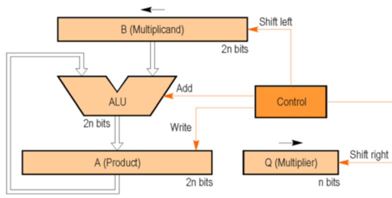
Positive Numbers
- Comp = original (补码=原码)
- Original = comp (原码=补码)

Negative Numbers
- **Comp = reverse(abs(original)) + 1 (补码=正数的反码+1)**
- **Original = -reverse(comp-1) (原码=补码-1/反转/再加负号)**

Negative Numbers
1. **统一加减法**：减法可以转换为加法来处理。
2. **简化溢出处理**：正负溢出方式相同，简化溢出的检测逻辑。
3. **避免两个零**：假若不用补码，则0000_2, 1000_2 都代表 0

Overflow detection
- Positive + Positive = Negative
- Negative + Negative = Positive

Multiplication



1. Left-shift multiplicand B
2. Right shift multiplier Q (so each time we can take Q[0])
3. A (product) = A + B * Q[0]

Floating Point Notation

- Significand: Normalized to 1 ($\pm 1.bbbb \ldots b \times 2^E$)
- Bias = 2^(k-1)-1, k is the number of bits of the "biased exponent"
- Bias Exponent: $\text{Biased exponent} = E + \text{bias}$

Decimal Fraction -> Binary Float Point:

−29.21875 → 32-bit IEEE floating-point

$$
\begin{aligned}
(29)_{10} &= 2 \cdot 14 \text{ ......}1 \\
14 &= 2 \cdot 7 \text{ ......}0 \\
7 &= 2 \cdot 3 \text{ ......}1 \\
3 &= 2 \cdot 1 \text{ ......}1 \\
1 &= 2 \cdot 0 \text{ ......}1 \\
&= (11101)_2
\end{aligned}
\qquad
\begin{aligned}
0.21875 \times 2 &= 0.4375\ (0) \\
0.4375 \times 2 &= 0.875\ (0) \\
0.875 \times 2 &= 1.75\ (1) \\
0.75 \times 2 &= 1.5\ (1) \\
0.5 \times 2 &= 1.0\ (1)
\end{aligned}
$$

Therefore $-29.21875 = -11101.0011 = -1.11010011 \times 2^4$

$$\text{Normalize:}\ -1.11010011 \times 2^4$$

$$\text{Bias exponent: } 4 + (2^{5-1}) = (19)_{10} \rightarrow (10011)_2$$

IEEE Floating Point Representation:

$$\text{IEEE Float Point} = 1\ 01111111\ 11010011000000000000000$$

MIPS

	Instruction	RTL	Notes
00	sll \$rd, \$rt, shamt	R[\$rd] ← R[\$rt] << shamt	
02	srl \$rd, \$rt, shamt	R[\$rd] ← R[\$rt] >> shamt	Unsigned right shift
03	sra \$rd, \$rt, shamt	R[\$rd] ← R[\$rt] >> shamt	Signed right shift
04	sllv \$rd, \$rt, \$rs	R[\$rd] ← R[\$rt] << R[\$rs]	
06	srlv \$rd, \$rt, \$rs	R[\$rd] ← R[\$rt] >> R[\$rs]	Unsigned right shift
07	srav \$rd, \$rt, \$rs	R[\$rd] ← R[\$rt] >> R[\$rs]	Signed right shift
08	jr \$rs	PC ← R[\$rs]	R[\$rs] must be a multiple of 4
09	jalr \$rd, \$rs	tmp ← R[\$rs] R[\$rd] ← PC + 8 PC ← tmp	R[\$rs] must be a multiple of 4;
09	jalr \$rs	(special form of "jalr \$rd, \$rs" where \$rd = 31, implicitly)	
12	syscall	System Call	
16	mfhi \$rd	R[\$rd] ← HI	
17	mthi \$rs	HI ← R[\$rs]	
18	mflo \$rd	R[\$rd] ← LO	
19	mtlo \$rs	LO ← R[\$rs]	
24	mult \$rs, \$rt	{HI, LO} ← R[\$rs] * R[\$rt]	Signed multiplication
25	multu \$rs, \$rt	{HI, LO} ← R[\$rs] * R[\$rt]	Unsigned multiplication
26	div \$rs, \$rt	LO ← R[\$rs] / R[\$rt] HI ← R[\$rs] % R[\$rt]	Signed division
27	divu \$rs, \$rt	LO ← R[\$rs] / R[\$rt] HI ← R[\$rs] % R[\$rt]	Unsigned division
32	add \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] + R[\$rt]	Exception on signed overflow
33	addu \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] + R[\$rt]	
34	sub \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] - R[\$rt]	Exception on signed overflow
35	subu \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] - R[\$rt]	
36	and \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] & R[\$rt]	
37	or \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] | R[\$rt]	
38	xor \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] ^ R[\$rt]	
39	nor \$rd, \$rs, \$rt	R[\$rd] ← !(R[\$rs] | R[\$rt])	
42	slt \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] < R[\$rt]	Signed comparison
43	sltu \$rd, \$rs, \$rt	R[\$rd] ← R[\$rs] < R[\$rt]	Unsigned comparison

Type	31 26	25 21	20 16	15 11	10 06	05 00
R-Type	opcode	\$rs	\$rt	\$rd	shamt	funct
I-Type	opcode	\$rs	\$rt	imm		
J-Type	opcode	address				

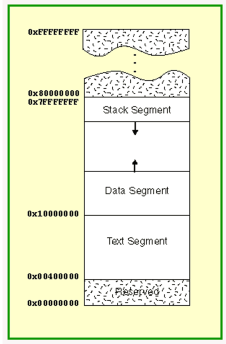
	Instruction	RTL
02	j address	PC ← {(PC + 4)[31:28], address, 00}
03	jal address	R[31] ← PC + 8 PC ← {(PC + 4)[31:28], address, 00}

Register Number	Conventional Name	Usage
\$0	\$zero	Hard-wired to 0
\$1	\$at	Reserved for pseudo-instructions
\$2 - \$3	\$v0, \$v1	Return values from functions
\$4 - \$7	\$a0 - \$a3	Arguments to functions - not preserved by subprograms
\$8 - \$15	\$t0 - \$t7	Temporary data, not preserved by subprograms
\$16 - \$23	\$s0 - \$s7	Saved registers, preserved by subprograms
\$24 - \$25	\$t8 - \$t9	More temporary registers, not preserved by subprograms
\$26 - \$27	\$k0 - \$k1	Reserved for kernel. Do not use.
\$28	\$gp	Global Area Pointer (base of global data segment)
\$29	\$sp	Stack Pointer
\$30	\$fp	Frame Pointer
\$31	\$ra	Return Address

J-type instructions
- Only 26bits address
- [leftmost 4 bits of PC] + [26bit address] + [00, 2 zeros]

I-type instructions
- The immediate value "imm" only have 16 bits
- Zero extend leftwards to extend it to 32 bits

R-type instructions
- \$rs = \$rt + \$rd. 5 Bit each register (since we have 32 in total)
- Shamt used in shift function



MIPS Instruction Cautions & Notes

Bit-wise operations
- ORI: Must be **non-negative** (2's comp. meaningless after zero extend). ORI with zero can load **non-negative** constant to register
- ANDI: **sign irrelevant** (AND with zero padding always be 0). Good for clearing registers.
- XORI: **sign irrelevant**. Good for flipping specific bits of a register (by setting some bits of the immediate value to 1)
- NOR: equivalent to **NOT** (anything NOR 0, flips its bit)
- SLL: implement **No-Op** (sll \$0, \$0, 0)
- SRL: must be **non-negative** (due to zero padding on the left)
- SRA: **sign extend** left (extend with leftmost bit)

Arithmetic operations
- ADDU: **NOT unsigned**. It means **overflow is ignored**
- ADD: **overflow is detected**
- ADDIU: overflow ignore, **sign extended**
- Same for SUBU / SUB (overflow ignored / detected)
- MULTU: multiply unsigned (no overflow check)
- MULT: multiply signed (no overflow check)
- MFHI D: move 'hi' register -> 'D' register
- MFLO D: move 'lo' register -> 'D' register
- DIVU: division for unsigned. **Remainder -> hi ; Quotient -> lo**

Memory Access Operations
- LW \$1, imm(\$2): \$1 <- RAM [\$2 + SignExtend(imm)]
- SW \$1, imm(\$2): RAM[\$2 + SignExtend(imm)] <- \$1
- LUI \$1, imm: copy imm to upper 16bits of \$1
- ORI \$1, \$0, imm: copy imm to lower 16bits of \$1
- .data Directive: Declare variables & constants. Data section start at 0x10000000.
- .word Directive: Allocates a 4Bytes. (.data 69 / .data 1, 2, 3, 4)

Control Flow Operations
- J / JAL, PC + 4 before jump, thus need Branch Delay SLot

Name	Instruction	Operation	Notes
Branch on $=$	beq \$rs, \$rt, imm	$\text{if}(R[\$\text{rs}] = R[\$\text{rt}])\text{PC} \leftarrow \text{PC} + 4 + \text{SignExt}_{18b}(\{\text{imm}, 00\})$	
Branch on $! =$	bne \$rs, \$rt, imm	$\text{if}(R[\$\text{rs}] \neq R[\$\text{rt}])\text{PC} \leftarrow \text{PC} + 4 + \text{SignExt}_{18b}(\{\text{imm}, 00\})$	
Branch on ≤ 0	blez \$rs, imm	$\text{if}(R[\$\text{rs}] \leq 0)\text{PC} \leftarrow \text{PC} + 4 + \text{SignExt}_{18b}(\{\text{imm}, 00\})$	Signed comparison
Branch on > 0	bgtz \$rs, imm	$\text{if}(R[\$\text{rs}] > 0)\text{PC} \leftarrow \text{PC} + 4 + \text{SignExt}_{18b}(\{\text{imm}, 00\})$	Signed comparison
Branch on < 0	bltz \$rs, imm	$\text{if}(R[\$\text{rs}] < 0)\text{PC} \leftarrow \text{PC} + 4 + \text{SignExt}_{18b}(\{\text{imm}, 00\})$	Signed Comparison
Branch on ≥ 0	bgez \$rs, imm	$\text{if}(R[\$\text{rs}] \geq 0)\text{PC} \leftarrow \text{PC} + 4 + \text{SignExt}_{18b}(\{\text{imm}, 00\})$	Signed Comparison
Set less than	slt \$rd, \$rs, \$rt	$R[\$\text{rd}] \leftarrow R[\$\text{rs}] < R[\$\text{rt}]$	Signed comparison
Set $<$ unsigned	sltu \$rd, \$rs, \$rt	$R[\$\text{rd}] \leftarrow R[\$\text{rs}] < R[\$\text{rt}]$	Unsigned comparison
Set $<$ immediate	slti \$rt, \$rs, imm	$R[\$\text{rt}] \leftarrow R[\$\text{rs}] < \text{SignExt}_{16b}(\text{imm})$	Signed comparison
Set $<$ imm unsigned	sltiu \$rt, \$rs, imm	$R[\$\text{rt}] \leftarrow R[\$\text{rs}] < \text{SignExt}_{16b}(\text{imm})$	Unsigned comparison

Subroutine
- \$ra (return address) <- PC + 8
- \$sp (stack pointer): points to stack top (smallest address)
- Stack push: subu \$sp, \$sp, 4 -> sw \$t0, (\$sp) --- (\$sp) equiv 0(\$sp)
- Stack pop: lw \$t0, (\$sp) -> addu \$sp, \$sp, 4

Boolean Algebra

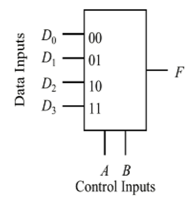
Number of Possible Switching Functions: 2^{2^N} (N input variables)

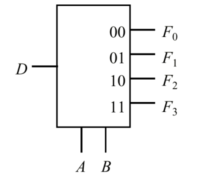
Commutative Laws	$A + B = B + A$	$A.B = B.A$
Associative Laws	$A + (B + C) = (A + B) + C$	$A.(B.C) = (A.B).C$
Distributive Laws	$A.(B + C) = (A.B) + (A.C)$ $(A + B).(A + C) = A + B.C$	$A + (B.C) = (A + B).(A + C)$
Tautology/Idempotent Laws	$A.A = A$	$A + A = A$
Tautology/Identity Laws	$1.A = A$	$0 + A = A$
Tautology/Null Laws	$0.A = 0$	$1 + A = 1$
Tautology/Inverse Laws	$A.\overline{A} = 0$	$A + \overline{A} = 1$
Absorption Laws	$A.(A + B) = A$ $A + A.B = A$	$A + (A.B) = A$ $A + \overline{A}.B = A + B$
De Morgan's Laws	$\overline{(A.B)} = \overline{A} + \overline{B}$	$\overline{(A + B)} = \overline{A}.\overline{B}$

Boolean Function Duality **(Right Side of Above Table)**
- Replace AND with OR ; Replace OR with AND
- Replace constant 1 with 0 ; Replace 0 with 1
- New function has exact opposite output.

Combinational Logic

Multiplexer (MUX) reduce wire usage **Demultiplexer (DeMUX)**





A	B	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$F_0 = D\overline{A}\,\overline{B}$ $\quad F_2 = DA\overline{B}$

$F_1 = D\overline{A}B$ $\quad F_3 = DAB$

D	A	B	F_0	F_1	F_2	F_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Karnaugh Map (K-Map)
- Bit order: 00 01 11 10 (both row and column)
- Group biggest blocks of 1s.
- "X" or "don't care" don't necessarily need to be circled. Only circle them when it simplifies things

Minterm: Product term in SOP form $\sum(m_i, m_j, \ldots)$

Maxterm: Sum term in POS form $\prod(M_i, M_j, \ldots)$

Quine-McKluskey tables
- Idea: if 2 minterms differ **only in one position**, can be simplified
- AB + AB' = A(B + B') = A

Procedure
1. Group minterms according to **the number of 1's**
2. Check adjacent groups for simplification (group 1 & 2, 2 & 3 …)
3. Merge minterms only differ by one position. Denote the differing position with '-' (canceled)
4. Repeated terms can be cancelled

Group Name	Min terms	W	X	Y	Z
GB1	2,6	0	-	1	0
	2,10	-	0	1	0
	8,9	1	0	0	-
	8,10	1	0	-	0
GB2	6,14	-	1	1	0
	9,11	1	0	-	1
	10,11	1	0	1	-
	10,14	1	-	1	0
GB3	11,15	1	-	1	1
	14,15	1	1	1	-

Group Name	Min terms	W	X	Y	Z
GB1	2,6,10,14	-	-	1	0
	2,10,6,14	-	-	1	0
	8,9,10,11	1	0	-	-
	8,10,9,11	1	0	-	-
GB2	10,11,14,15	1	-	1	-
	10,14,11,15	1	-	1	-

5. Repeat 1~5, Until Can't merge. Now we get **prime implicant**: a minterm that cannot be further simplified with other minterms

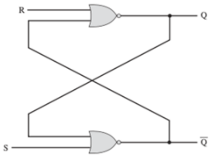
Prime Implicant Table: each prime implicant covers some minterms

Min terms / Prime Implicants	2	6	8	9	10	11	14	15
YZ'	1	1			1		1	
WX'			1	1	1	1		
WY					1	1	1	1

上面表格中如有 minterm **仅被 1 个 prime implicant 覆盖**，那么这个 prime implicant 是 **essential prime implicant**

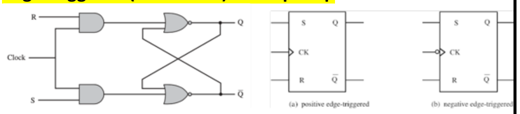
Sequential Circuit
- Not only dependent on current (comb. circuits) but also past behavior (i.e., storage element / memory)
- Output (next state) = current input + current state.
- Flip-flops: **bistable** device (two stable states – doesn't change without external input). Outputs: $Q, \overline{Q}$ complements each other.

SR Flipflops

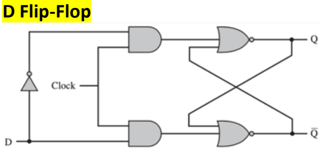


S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	–

Edge Triggered (or Clocked) S-R Flip-Flop



D Flip-Flop

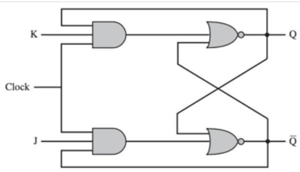


D	Q_{n+1}
0	0
1	1

Characteristic table

JK Flip-Flop

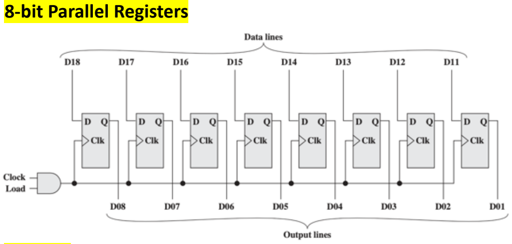
- Unlike SR Flip-Flop (which can't take S=1, R=1 as input)
- JK Flip-Flop allows all combination of input (including 1 1)
- Output tick between 1, 0 each clock cycle when J=1 K=1



J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

Characteristic table

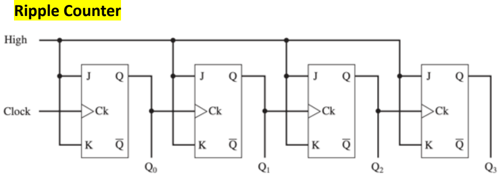
8-bit Parallel Registers



Counter (register whose value incr. by 1 every tick)

- For counter with N flip flops, the value ranges from 0 ~ $2^N - 1$
- Asynchronous counter: state of the FF will NOT change same time
- Synchronous counter: state of the FF changes at the same time

Ripple Counter



Finite State Machine

Definition

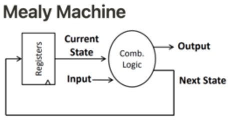
Finite State Machine (FSM): abstract model to describe real world systems

- States: represent different situation of the system
- FSM models **state changes** (external input → external output relation)

$$FSM = (S, I, O, \pi)$$

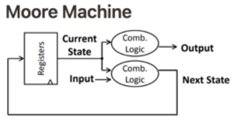
- S: the finite set of states
- I: the finite set of external inputs
- O: the finite set of external outputs
- π: state transition function:
  - Define the **relations** among input, output, current state, next state.
  - **Complete:** for any $\{S_t, I_t\}$, $\{S_{t+1}, I_{t+1}\}$ can be determined

Mealy Machine



- Output and state depends on both current state and input

Moore Machine



- Output only depends on current state

- **Registers = Memory Component**
- **Comb. logic = State transition Function**

FSM Model -> Circuit

- Model the problem using FSM
- Construct **excitation table** of the circuit

Columns:

- Q: current state
- X: Input
- Q_{next}: next state
- Input to Flip-Flop

Basically:

- FSM -> State table
- FF -> FF ExTable
- **Circuit table** = FF ExTable + State Table

CHARACTERISTIC EQUATION	EXCITATION TABLE			
$Q_{(next)} = S + R'Q$, $SR = 0$	Q	Q(next)	S	R
	0	0	0	X
	0	1	1	0
	1	0	0	1
	1	1	X	0
$Q_{(next)} = JQ' + K'Q$	Q	Q(next)	J	K
	0	0	0	X
	0	1	1	X
	1	0	X	1
	1	1	X	0
$Q_{(next)} = D$	Q	Q(next)	D	
	0	0	0	
	0	1	1	
	1	0	0	
	1	1	1	
$Q_{(next)} = TQ' + T'Q$	Q	Q(next)	T	
	0	0	0	
	0	1	1	
	1	0	1	
	1	1	0	

- K-map simplification: All pairs of **(State Q) vs (Flip-Flop Input)**

Cache Mapping

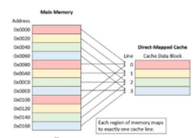
Mapping Function

Intuition:

- $\because C \ll M$ (significantly more memory blocks than cachelines)
- $\therefore$ Use mapping function to map "blocks → cacheline"
- $\text{line} = f(\text{block})$

Direct Mapping

- **Explanation:** Multiple blocks of memory mapped to one fixed cacheline
- **Mapping Rule:**
  - Q blocks mapped to 1 cacheline.
  - $Q = \frac{M}{C}$
- **Selecting cacheline:**
  - $\text{index} = \text{block address} \mod C$
- **Cons:** low hit ratio if accesses clusters



Associative Mapping

- **Explanation:** Any block can map to any cacheline
- **Mapping Rule:**
  - If there's spare lines, map there
  - Search tags through cached blocks sequentially or in parallel.
- **Cons:** Checking tags is complex

Cache Performance

- Miss Rate = 1 – Hit Rate (*% of mem accesses not found in cache)*
- Hit Time = Time to deliver word from cache -> memory
- Miss Penalty = extra time needed (to access memory) due to miss

Memory Address

Block Address		Block Offset
Tag	Index	

- **Block Offset:** index of each word within the block.
- **Index:** indicates which cacheline
- **Tag:** unique identifier of blocks mapped to same cacheline (blocks of same cacheline have ***different tags***)

Calculations:

- N bit address -> $\boldsymbol{2^n}$ **addressable words**
- $\boldsymbol{K}$ words per block -> $\boldsymbol{M = 2^n/K}$ **blocks**
- **Offset**: $log_2(K)$ bits --- **log (Block Size) bits**
- **Index:** $log_2(C)$ bits --- **log (Num. of Cacheline) bits**
- **Tag:** $\boldsymbol{n - log_2(K) - log_2(C)}$ --- Total Length of Address – Length of Offset – Length of Index
-

Memory Access Logic

- Row Access: 11 lines + RAS -> select 2048 rows
- Col Access: 11 lines + CAS -> select 2048 columns
- Read/write: controlled by WE / OE pins.

Error Detection and Correction

- Codeword = original data + parity bits
- D_min = minimum hamming distance among codewords
- **Max Detectable Error: D_min – 1 bits**
- **Max Correctable Error: Floor ((D_min – 1) / 2) bits**

Contiguous Memory Allocation

Fixed Partition

- Memory divided into one or more fixed-sized pattern

Variable Parition

- Division not fixed size (process allocated **just enough** memory), thus no internal fragmentation

Fragmentation

- Internal: unused space **within** partition (fixed partition only)
- External: enough total space, but non-contiguous
- Solution: compaction (defragmentation)

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level		
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

Non-Contiguous Memory Allocation

Paging

- **Page Table:** translate page -> Frames
- **Page:** logical memory divided into fixed-size blocks (process view)
- **Frame:** physical mem divided into fixed-size blocks (actual)
- **Virtual Address = { Page Number, Page Offset }**
- Page Offset has log_2(page & frame size)

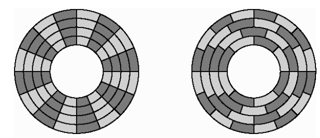
Segmentation

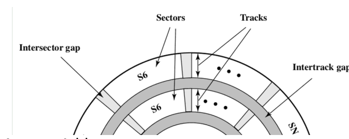
- Program divided into variable-szied segments
- **Virtual Address = { Segment Number, Segment Offset }**

- **Segment Table:**
- Base Address : Physical starting address of the segment
- Segment Limit: Length of that segment
- Physical Address = Base Address + Segment Offset

Magnetic Disk

- Constant Angular Velocity (CAV) – **LEFT**
- Multiple Zoned Recording -- **RIGHT**





- Access Must Specify: Cylinder, Head, Sector, Transfer Size, Memory Address
- **Access Delay = Seek Time + Rotational Delay**
- **Format of a Track** [Sector] [Sector] [Sector] …
- **Format of Sector** [gap] [ID field] [gap] [Data Field] [gap]
- **ID Field:** [Synth Byte] [Track] [Head] [Sector] [CRC]

Solid-State Drive (SSD)

- Erase entire block before write
- Write data in new space then garbage collect 'deleted' space

I/O Modules - Function

- **Control and Timing**: Proc request; I/OM check & return device status; transfer data if device ready
- **Processor Communication: Command** (control bus, CPU -> I/OM), **Data** (data bus), **Status** (BUSY, READY, error conditions)
- **Device Communication: I/O -> devices.** Unique address for each device (depending on Memory Mapped I/O or Isolated I/O)
- **Data Buffering:** I/OM buffers data to handle different transfer rates.
- **Error Detection:** soft error (parity bit) OR hard error (paper jam)

I/O Modules – Technique

Programmed I/O (PIO): I/O Device -> CPU -> RAM

- CPU -> IOM **sends command**
- IOM perform action, sets **status register** when finished
- **Pooling:** CPU periodically checks status register to check completion

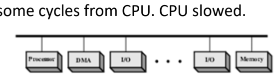
Interrupt-Driven I/O: I/O Device -> CPU -> RAM

- Same as PIO. Except after command send, CPU **continue other work**
- IOM perform action, **interrupt** CPU when ready
- CPU save state (register, PC, etc), process interrupt, restore state.

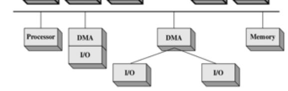
Direct Memory Access (DMA): I/O device -> DMA -> RAM

- CPU tells DMA what to do (e.g., device addr, memory loc, words)
- DMA transfer entire block of data. Interrupt CPU when finished.
- **Cycle Stealing:** DMA *steals* some cycles from CPU. CPU slowed.

- **Single bus, detached DMA.** Use bus twice
- CPU suspend twice



- **Single bus, integrated DMA.** Use bus once
- CPU suspend once



- **Separate I/O Bus.**
- CPU suspended once
- Use Bus once