0x983412AB be stored in Big-Endian Mode and Little-Endian Mode

Big-Endian Mode: (left to right)

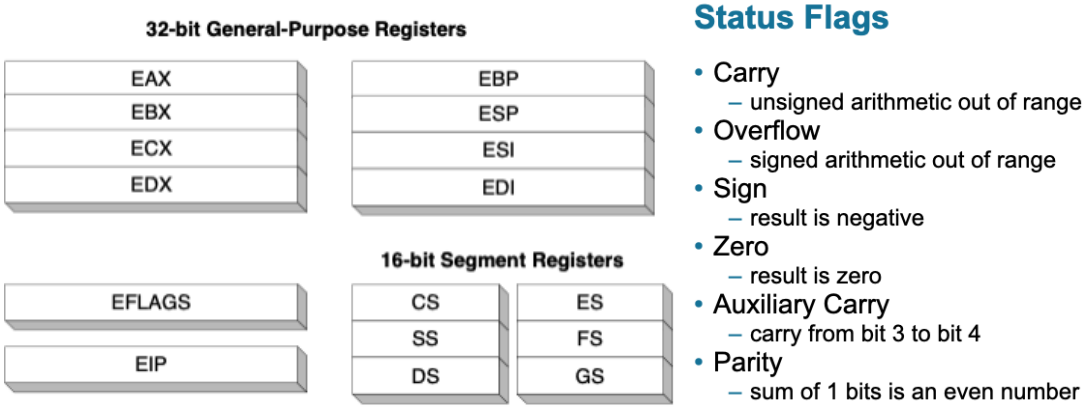
- Address: 0x1000 0x1001 0x1002 0x1003
- Value: 0x98 0x34 0x12 0xAB

Little-Endian Mode: (right to left)

- Address: 0x1000 0x1001 0x1002 0x1003
- Value: 0xAB 0x12 0x34 0x98

Intel

- Instruction Execution Cycle: Fetch, Decode, Fetch Operands, Execute, Store Output
- Cache hit: when data to be read is already in cache memory
- Cache miss: when data to be read isn't in cache memory



- Address mapping: convert address in a set of instructions into an absolute address

Memory

- Dynamic RAM(DRAM): inexpensive, must be refreshed constantly
- Static RAM(SRAM): expensive, used for cache memory, no refresh required
- Video RAM(VRAM): dual ported, optimized for constant video refresh
- CMOS RAM: complimentary metal-oxide semiconductor, system setup information

Levels of Input-Output

- Level 3: high-level language function. Ex: C++, Java
- Level 2: Operating System. Ex: API
- Level 1: BIOS. Driver that communicates directly with devices

Assembly Language

- MOV destination, source // move from source to destination
- MOVZX destination, source //when copy smaller value into larger destination, fills the upper half of the destination with zeros
- MOVSX destination, source // fills the upper half of the destination with the source's operand sign bit
- XCHG destination, source // exchanges the values of two operands
- INC destination //destination = destination + 1

- DEC destination //destination = destination - 1
- ADD destination, source // destination = destination + source
- SUB destination, source // destination = destination - source
- NEG destination, source // reverse the sign of an operand, positive to negative
- OFFSET value // returns the distance in bytes
- TYPE value // returns the size in bytes of the element
- LENGTHOF // counts the number of elements in data declaration
- SIZEOF // returns a value that is equivalent to LENGTHOF * TYPE
- JMP target // unconditional jump to a label of the same procedure
- LOOP target // creates a counting loop
- AND destination, source // performs boolean AND operation between matching bits of two operands
- OR destination, source // performs boolean OR operation between matching bits of two operands
- XOR destination, source // performs boolean exclusive-OR operations between matching bits of two operands
- NOT destination // performs boolean NOT operation on single destination operand
- CMP destination, source // compares the destination operand to the source operand
    - Destination == source // zero flag set
    - Destination < source // carry flag set
    - Destination > source // zero and carry flag are cleared

- Jcond // conditional jump instruction to label when flag condition are met

Jumps Based on Specific Flags

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

-

Flags:

- Zero flag: set when destination equals to zero

- Sign flag: set when destination is negative
- Carry flag: set when unsigned value is out of range
- Overflow flag: set when signed value is out of range

Stack

- LIFO(Last in first out)
- PUSH operation // decrements the stack pointer by 4 and copies a value into the location pointed by the stack pointer
- POP operation // copies value at stack into a register/variable

Memory

- Dynamic RAM(DRAM): refresh every few milliseconds, “cheap”,
- Static RAM(SDAM): similar to D flip-flop, fast, build cache memory,
- ROM: doesn’t need to be refreshed, store permanent
- Principle of locality: once a byte is accessed, the nearby data element would be needed soon
  - Temporal locality: Recently-accessed data elements tend to be accessed again.
  - Spatial locality: Accesses tend to cluster.
  - Sequential locality: Instructions tend to be accessed sequentially.
- Cache memory: speed up access by storing used data closer to CPU, instead of storing in main memory. Cache is accessed by content
- Offset field: uniquely identifies an address within a specific block.
- Block field: selects a unique block of cache.
- Tag field: is whatever is left over.

- EAT = H ◊ $Access_C$ + (1 − H) ◊ $Access_{MM}$
  - where H is the cache hit rate and $Access_C$ and $Access_{MM}$ are the access times for cache and main memory, respectively.
  - For example, consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.
  - 0.99(10ns) + 0.01(200ns) = 9.9ns + 2ns = 11ns
- Write through: updates cache and main memory simultaneously on every write
- Write back: updates memory only when the block is selected for replacement
- Harvard cache: separate caches for data and instructions
- Strictly inclusive: cache guarantee that all data in smaller cache also exist at the next higher level
- Exclusive: permits only one copy of the data

- Physical address: the actual memory address of physical memory
- Virtual addresses: programs creation that mapped to physical addressed from disk
- Page faults: when logical address requires that a page be brought in from disk
- Memory fragmentation: when paging process results in creation of small, unusable clusters of memory addressed.
- Page field: determines the page location of the address
- Offset field: indicates the location of the address within the page
- Amdahl's law: the overall performance of a system is a result of the interaction of all of its components. System performance is most effective when its used components are improved
    - $S = \frac{1}{(1-f)+(f/k)}$ where S is the overall speedup;
    - f is the fraction of work performed by a faster component; and k is the speedup of the faster component.

- Programmed I/O: reserves a register for each I/O device, each register is continually polled to detect data arrival
- Interrupt-Driven I/O: allows the CPU to do other things until I/O is requested
- Memory-Mapped I/O: shares memory address space between I/O devices and program memory
- Direct Memory Access: offloads I/O processing to a special purpose chip that takes care of the details
- Channel I/O: dedicated I/O processors
- Character I/O: process one byte at a time. Ex: modems, keyboard, mice
- Block I/O: handle bytes in groups. Ex: disk, tape
- Width: the number of data lines in the bus
- Bus clock: coordinates activities and provides bit cell boundaries

Rigid Disk Drives

- Seek time: the time it takes for the disk arm to move into position over the desired cylinder
- Rotational delay: the time that it takes for the desired sector to move into position beneath the read/write head
- Access time: seek time + rotational delay + transfer time
- Transfer rate: the rate at which data can be read from the disk
- Average latency: function of the rotational speed

RAID

- RAID, Redundant Array of Independent Disks, addressed the problems of disk reliability, cost, and performance. Data is stored across many disks, with extra disks added to the array to provide error correction
- RAID Level 0: aka drive spanning, provides improved performance, but no redundancy, data is written in blocks across the entire array, low reliability.**optimal performance, no redundancy**
- RAID Level 1: disk mirroring, provides 100% redundancy & good performance. Two matched sets of disks contain the same data, high cost. **great performance, total redundancy, high cost**
- RAID Level 2: consists of sets of data drives, and set of hamming code drives. Hamming code provides error correction for the data drives. Poor performance and high cost
- RAID Level 3: strips bits across a set of data drives and provides a separate disk for parity. Parity is the XOR of the data bits. Not suitable for commercial applications, but great for personal systems.
- RAID Level 4: like adding parity disks to RAID 0. Data is written in blocks across the data disks, and a parity block is written to the redundant drive. Feasible if all record blocks were the same size.
- RAID Level 5: RAID 4 with distributed parity. Some accesses can be serviced concurrently, good performance and high reliability. Used in many commercial systems
- RAID Level 6: two levels of error protection over striped data. Can tolerate the loss of two disks. Write-intensive, but highly fault-tolerant

System Softwares

- Multiprogramming: allocating each process a given portion of CPU time
- Long-term scheduling: operating system determines which process shall be granted access to the CPU
- Short-term scheduling: operating system determines which one will have access to the CPU at a given moment
- Context switch: when a process is taken from the CPU and replaced by another process
- Binding: process of assigning physical addresses to program variables
- Dynamic linking: when the link editing is delayed until load/run time

Alternative Architectures

- Flynn's Taxonomy: takes into consideration the number of processors and number of data paths incorporated into an architecture. Strategy to categorize computer architectures

    - SISD: Single instruction stream, single data stream. These are classic uniprocessor systems.
    - SIMD: Single instruction stream, multiple data streams. Execute the same instruction on multiple data values, as in vector processors.
    - MIMD: Multiple instruction streams, multiple data streams. These are today's parallel architectures.
    - MISD: Multiple instruction streams, single data streams.
- FINAL REVIEW
    - TCP(transmission control protocol): connection-based protocol, reliable, transfer data slowly. Once connection is established, data can be sent bidirectionally
    - UDP(user datagram protocol): connectionless, less-reliable, quicker, multiple messages are sent as packets in chunks using UDP
    - TCP sequence number in TCP windowing: offset 32 into TCP header is the sequence number. Sequence number is a counter used to keep track of every byte sent outward by host
    - IP address: unique address that identifies a device on the internet/network
    - Bytecode in Java: set of instructions for the Java Virtual Machine. Code that lies between the low-level and high-level language. After the java code is compiled, the bytecode gets generated, which can be executed on any machine using JVM
    - Garbage Collection: automated process of deleting code no longer needed/used. Frees memory space, makes coding Java apps easier
    - Quantum Computing: tech that harnesses the laws of quantum mechanics to solve problems too complex for classical computers.
    - Quantum mechanics: description of the physical properties of nature at the scale of atoms and subatomic particles.
    - Qubit: quantum bit, basic unit of quantum information. Two state quantum-mechanical system
    - Decoherence: non-unitary process by which a system couples with its environments. Describes the disappearance/absence of certain superpositions of quantum states
    - Compiler optimization: compiler minimize/maximize attributes of executable computer program
    - Static library: resolved in caller in compiled-time and copied into a target application by the compiler. Share libraries get import at the time of execution of target program by the operating system itself

Copying a String

The following code copies a string from source to target:

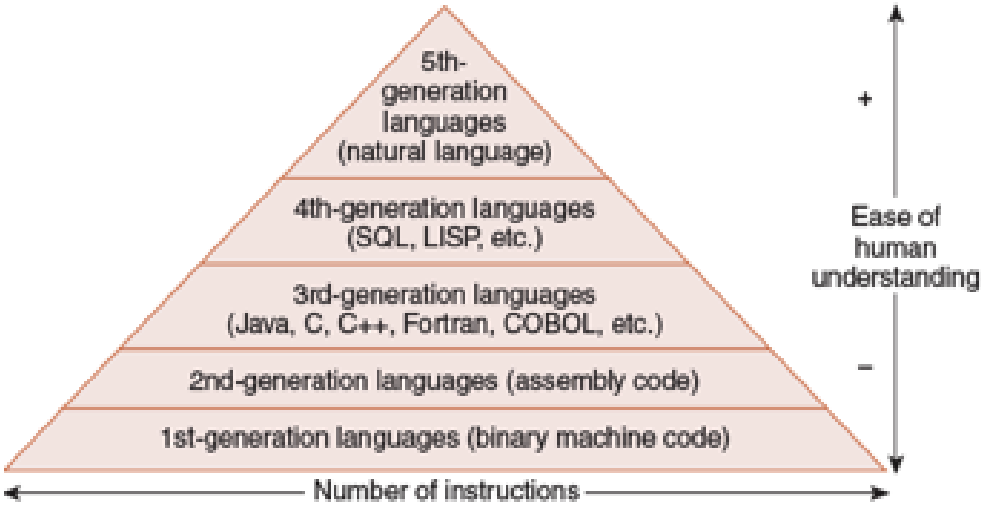
```
.data
source BYTE "This is the source string",0
target  BYTE SIZEOF source DUP(0)

.code
      mov esi,0                  ; index register
      mov ecx,SIZEOF source      ; loop counter
L1:
      mov al,source[esi]         ; get char from source
      mov target[esi],al         ; store it in the target
      inc  esi                   ; move to next character
      loop L1                    ; repeat for entire string
```

good use of SIZEOF

- CISC
  - Single register set
  - One or two register operands per instruction
  - Parameter passing through memory
  - Multiple cycle instructions
  - Microprogrammed control
  - Less pipelined
- RISC
  - Multiple register sets
  - Three operands per instruction
  - Parameter passing through register windows
  - Single-cycle instructions
  - Hardwired control
  - Highly pipelined

- CISC
  - Many complex instructions
  - Variable length instructions
  - Complexity in microcode
  - Many instructions can access memory
  - Many addressing modes
- RISC
  - Simple instructions, few in number
  - Fixed length instructions
  - Complexity in compiler
  - Only **LOAD/STORE** instructions access memory
  - Few addressing modes



Summing an Integer Array

The following code calculates the sum of an array of 16-bit integers.

```
.data
intarray WORD 100h,200h,300h,400h
.code
    mov edi,OFFSET intarray        ; address of intarray
    mov ecx,LENGTHOF intarray      ; loop counter
    mov ax,0                       ; zero the accumulator
L1:
    add ax,[edi]                   ; add an integer
    add edi,TYPE intarray          ; point to next integer
    loop L1                        ; repeat until ECX = 0
```

```
// C++ version:

char array[1000];
char * p = array;
```

```
; Assembly language:

.data
array BYTE 1000 DUP(?)
.code
mov esi,OFFSET array
```

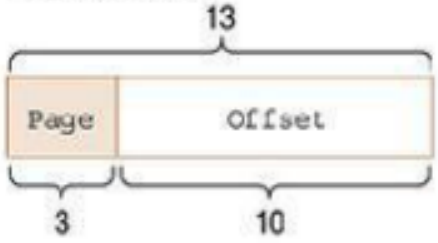
Nested Loop

If you need to code a loop within a loop, you must save the outer loop counter's ECX value. In the following example, the outer loop executes 100 times, and the inner loop 20 times.

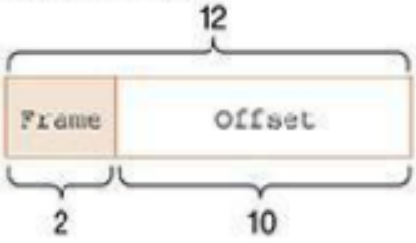
```
.data
count DWORD ?
.code
      mov ecx,100             ; set outer loop count
L1:
      mov count,ecx           ; save outer loop count
      mov ecx,20              ; set inner loop count
L2:   .
      .
      loop L2                 ; repeat the inner loop
      mov ecx,count           ; restore outer loop count
      loop L1                 ; repeat the outer loop
```

Virtual address space: 8K = 2^{13}
Physical memory: 4K = 2^{12}
Page size: 1K = 2^{10}

(a) Virtual address



Physical address



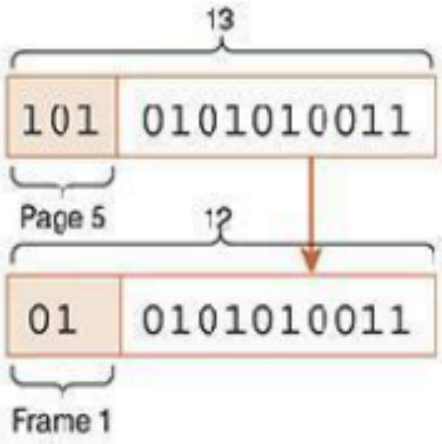
(b) Page table

Page	Frame	Valid bit
0	-	0
1	3	1
2	0	1
3	-	0
4	-	0
5	1	1
6	2	1
7	-	0

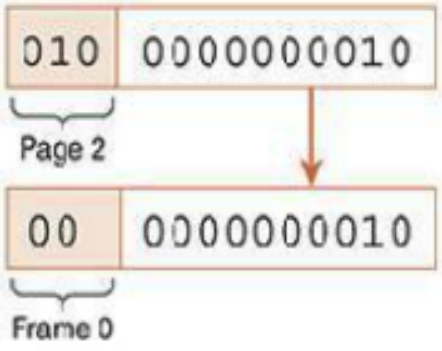
(c) Addresses

Page	Base 10	Base 16
0	0 - 1023	0 - 3FF
1	1024 - 2047	400 - 7FF
2	2048 - 3071	800 - BFF
3	3072 - 4095	C00 - FFF
4	4096 - 5119	1000 - 13FF
5	5120 - 6143	1400 - 17FF
6	6144 - 7167	1800 - 1BFF
7	7168 - 8191	1C00 - 1FFF

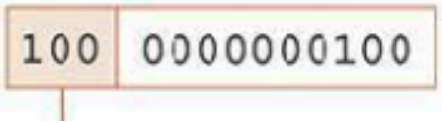
(d) Virtual address 0x1553 is converted to Physical address 0x553



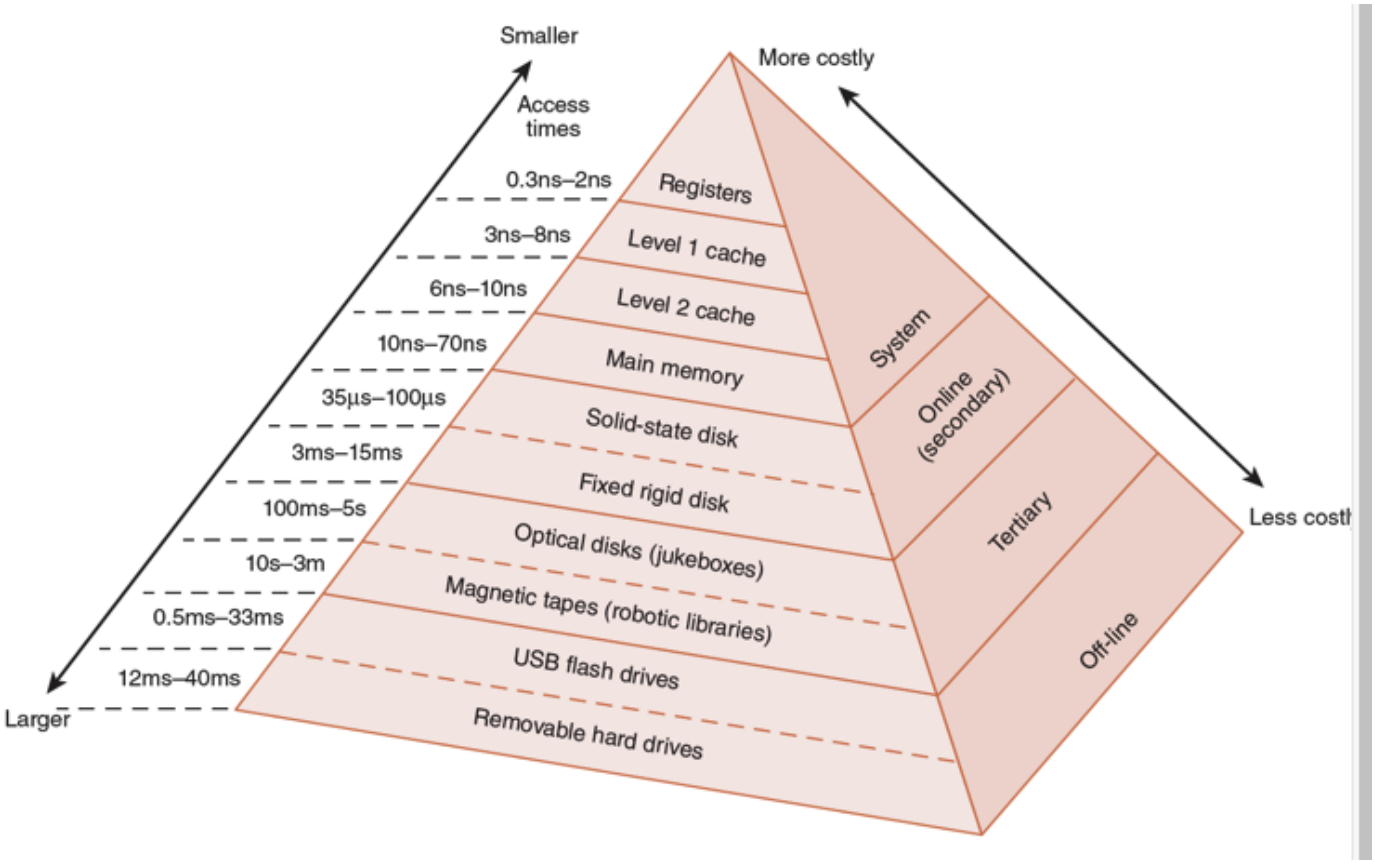
(e) Virtual address 0x802 is converted to Physical address 0x002



(f) Virtual address 0x1004



RISC	CISC
Multiple register sets, often consisting of more than 256 registers	Single register set, typically 6 to 16 registers total
Three register operands allowed per instruction (e.g., add R1, R2, R3)	One or two register operands allowed per instruction (e.g., add R1, R2)
Parameter passing through efficient on-chip register windows	Parameter passing through inefficient off-chip memory
Single-cycle instructions (except for load and store	Multiple-cycle instructions
Hardwired control	Microprogrammed control
Highly pipelined	Less pipelined
Simple instructions that are few in number	Many complex instructions
Fixed-length instructions	Variable-length instructions
Complexity in compiler	Complexity in microcode
Only load and store instructions can access memory	Many instructions can access memory
Few addressing modes	Many addressing modes



Examples:

```
-99 (10) to hex
99 / 2 = 49 r 1
49 / 2 = 24 r 1
24 / 2 = 12 r 0
12 / 2 = 6 r 0
6  / 2 = 3 r 0
3  / 2 = 1 r 1
1  / 2 = 0 r 1
0 1 1 0 0 0 1 1 = 63 (16)
1 0 0 1 1 1 0 0 = one complement
              +1
1 0 0 1 1 1 0 1 = 9D (16)
 1
 9A4C
+16B2
=====
 B0FE
 3210
11 * 16^3 + 0 * 16^2 + 15 * 16^1 + 14 * 16^0 = 45310 (10)
  9
 9A4C
-16B2
=====
 839A
```

Big Endian

```
00 00 B0 FE
```

Little Endian

```
FE B0 00 00
```

"789 a Σ

```
ASCII = 37 38 39 20 61 20 ??
Unicode =          0037 0038 0039 0020 0061 0020 03A3
Unicode (little) = 3700 3800 3900 2000 6100 2000 A303
```

int xx[3] = {1, 2, 3};

```
1              2            3
00 00 00 01    00 00 00 02  00 00 00 03 (big endian)
01 00 00 00    02 00 00 00  03 00 00 00 (little endian)
```

Cache:

Each addresses tag + directMap + offset

Offset bit = size of each cache lines

directMap size = size of cache

cache line = 16 bytes

cache size = 256 lines

address = 12345678_{16} = 12345 67 8

offset = 8_{16}

cache entry = 67_{16}

tag = 12345_{16}

Virtual Address:

Page number + offset

offset size = size of each page frame

page size = 4096

virtual address space = 4096 pages

physical address space = 1024 pages

```
Page table
0 - 01
1 - 0D
2 - 02
3 - N/A
4 - 04

Virtual Address =  XXX YYY16
                   001 01016 -> 0D 01016
                          002 0FF16 -> 02 0FF16
                          003 10016 -> page fault
```

Memory Layout:

address 0:

- code
- global data and constants
- heap (grows to higher addresses)
- ...
- stack (grows to lower addresses)

address N (highest)