← support XIP →

SRAM

- in system memory
- volatile
- Large (4 - 6 transistors per cell) → 1 bit
- Fast
- low power consumption (active & standby)
- does NOT need periodic refresh

able to access random address locations within the memory at any point in time

- memory array takes up most of the space

DRAM

- in system memory
- volatile
- small (1 - 3 transistors per cell)
- Slower
- Higher power consumption } cause of periodic refresh
- Periodic refresh (maintain data integrity)
- single transistor design

- uses capacitor as its storage element
- Upgraded version: SDRAM
  ⇒ makes use of a clock signal from host to synchronize data transfer → faster transfer rate
- perfect memory choice for system memory
- pipline architecture
  → allow faster, overlapping operations

EEPROM

- non-volatile
- can electrically program/erase device

NAND Flash

- in main storage memory
- non-volatile
- behaves like a NAND gate.
- does NOT support XIP
- lower cost per bit than NOR
- e.g. USB, SSD

- can be erased in larger block size than EEPROM
- faster than EEPROM (write operation for large data)
- cost less than EEPROM
- suitable for system requiring large amt of non-volatile memory (storage mem)

SSD based on flash
NAND preferenced

NOR Flash

- in system memory*
- non-volatile
- behaves like a NOR gate.
- Supports XIP
- allows random work/byte programming (erasure done at block level)
- use as system memory to store program code or general storage memory

Magnetic HDD

- non-volatile
- stores data by magnetizing a thin film of ferromagnetic media on the platter
- mainly on desktop, data centres, basically large secondary storage devices.

Parallel Data Transfer	
Pros	Cons
>> Faster data transfer rate (more bits can be transferred at one time; given same clock rate as serial)	>> Signal Skew & Cross Talk (limits max clk speed & transfer distance)
>> Hardware interface design tend to be simpler as only strobe signals are needed.	>> Bulky hardware if data width is large
	>> need more space to route the PCB traces
	>> higher hardware costs

Serial Data Transfer	
Pros	Cons
>> Less affected by Signal Skew & Cross talk (less wires → able to support higher frequency data)	>> Data transfer rate lower given same clk rate since only 1 data line available
>> Transfer data reliably over a longer distance	>> more complex hardware interface as it needs to handle serial to parallel conversion (Processors typically only process in bytes, etc.)
>> Lower cost (less wires, etc)	

Synchronous

>> has common clock signal

E.g. receive to latch in data at every rising edge

>> Master - Slave; Master provide clk signal

>> Allow faster transfer rate (no data overhead needed)

Asynchronous

>> no common clock signal; so devices have to agree on a pre-fixed clock frequency to use for data transfer

>> reciever knows transmitting clk rate & no. of bits to be transferred with each data pack.

>> transferred by frame; every frame has data bits + overhead

Burst Mode

>> Allow faster data transfer
>> CPU may be suspended for longer period of time
>> may not be suitable for real-time applications
>> Suitable for application where transfer is bursty
E.g. HDD file transfer

Cycle Stealing

>> Slower data transfer rate
>> Interleaving DMA data transfer with CPU instructions allow CPU to continue executing its program while DMAC is doing data transfer
>> CPU performance slower; more responsive than Burst
>> Suitable for real time applications

Transparent Mode

>> Slowest data transfer rate but best CPU respond
>> may not affect CPU performance at all
>> more complex hardware needed to detect when CPU not using bus

Polling

Pros	Cons
>> Programmer has complete control over entire process >> Easiest method to test & debug	>> CPU waits in loop → cannot perform other task until data transfer done >> Program execution of CPU held up while waiting for I/O device to get ready >> Inefficient use of CPU resources

Pros	Cons
>> Efficient use of CPU resources >> CPU can continue with other tasks between interrupts >> allows prioritization & pre-emption	>> More hardware >> more complex & difficult to debug

Signal Chain Sub-System

>> ADC - DAC

>> Parallel and Serial (UART / SPI) Digital Interfaces

>> Interrupt Controller

>> Direct Memory Access Controller

Analog & Digital Signals

>> Real world signals analog by nature

>> Analog : continuous voltage level & in time domain

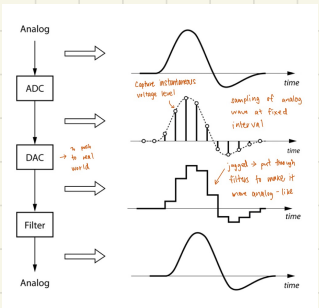
>> digital processing : more flexibility in implementation and are more tolerant to noise and component aging

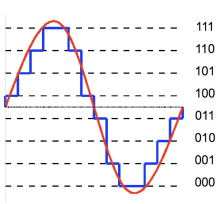
>> Digital signals are discrete time representation of analog signal ; has discrete voltage levels

>> Sampling frequency needs to be at least twice the signal frequency (Nyquist Theorem)

E.g. signal @ 1k hertz, sample @ 2K hertz / s

>> analog signal level is assigned to the nearest discrete voltage level allowed in process.





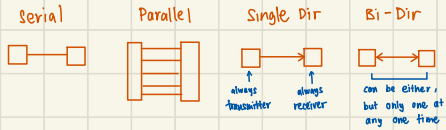
- Figure on the left shows a typical digital quantization process of a analog signal.
- A 3-bit system is used in this example. So all signal will be mapped to one of the 8 possible representations (000 to 111).
- At each sampling point, the analog signal level is approximated to the nearest digital equivalent.

- Digital Processors works only in the digital domain so typical process these days is to convert the real world analog signal to digital signal, allow the processor to work on the digital data, and reconstruct back the analog signal to be output to the real world.

Electrical Signal Interface

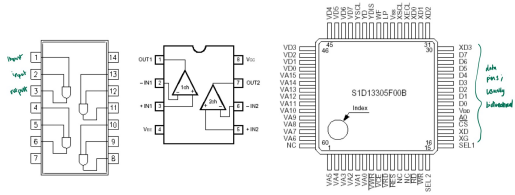
>> Interface : boundary wher >= 2 devices meet to exchange information

>> modes of connection : Single-bit (serial) / Multi-bit (Parallel) Data Transfer ; can be single or bi-direction.



Input / Output / Bi-Directional Pins

- Semiconductor devices interface to the outside world via pins.
- Depending on the device design and configuration, these pins are either an input, output or bi-direction (input and output) pin.



Interface Compatibility

>> Interfacing devices requires compatibility in :

Electrical signal level

Communication protocol (Handshaking & Data Signals)

Have to be the same

Electric signal level

>> if exceed max input voltage : spoilt / reduced reliability

>> output voltage should not exceed max. allowable input voltage

- Device #1
  - Output Voltage level for logic '1' = V1+ (e.g. 5V)
    For typical value of V1+, look for V(OH) parameter in device datasheets.
  - Output Voltage level for logic '0' = V1- (e.g. 0V)
    For typical value of V1-, look for V(OL) parameter in device datasheets.
- Device #2
  - Min Input voltage range to recognized as logic '1' = V(IH)
  - Max Input voltage range to recognized as logic '0' = V(IL)
- In order for Device #2 to sense the logic level correctly,
  - Condition 1: V1+ ≥ V(IH) (e.g. V1+ ≥ 2.0V)
  - Condition 2: V1- ≤ V(IL) (e.g. V1- ≤ 0.8V)

min. voltage device outputs for output logic 1

max. voltage device outputs for output logic 0

KEY:
VOH ≥ VIH
VOL ≤ VIL

Electrical Signal level Standards

>> VOH > VIH

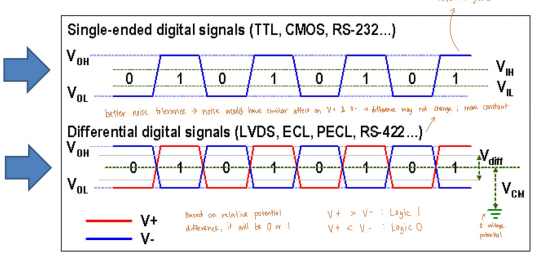
cater to potential drop in voltage when travelling across wires with resistance

>> VIL > VOL

cater to ground noise in electrical signal; to bump up signal

Differential Signals

- Differential signals has better noise tolerance so is able to be clocked at higher frequency.
- V(CM) = Common Mode Ground.



Communication Protocols

>> refers to how data are formatted during transmission

>> E.g. No. of bits in a transmission frame; what synchronization to use; Data width; Types of data and its formatting (USB, UART, SPI, etc.)

Parallel Data Transfer

>> Multiple bits of data transferred simultaneously between two devices

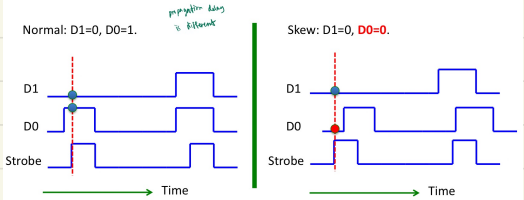
>> Synchronous in nature as some sort of strobe signal is needed to inform receiver when to latch data

>> higher transfer rate than serial (same clk rate)

>> more prone to Signal Skew and Crosstalk

Signal Skew

- If for some reason (due to circuit design, external interference), the signal in one or more data lines took different amount of time to reach the receiver, then there is a skew between the signals in the parallel data bus.
- Below example illustrate the effect of signal skew. Data is latched by the receiver on rising edge of the strobe signal.
- This result in wrong data (D0=0 instead of 1) being latched by the receiver.



Signal Skew – Contributing Factors

- Signal Skew is due to variation in propagation delay between signals from the same data bus.
- Propagation delay is the time taken for signal to travel between two points in a circuit.
- Capacitance and Resistance of the physical data line is a major contributor to circuit propagation delay.
  - The larger the resistance and capacitance, the larger the delay. Illustrated in the equation $\tau=RC$ where τ is the time constant dictating rate of change of voltage levels in the data line concerned.
- Variation in resistance and capacitance of the signal lines can be due to
  - Variation in PCB trace length/width (lead to change in impedance).
  - Connecting active components (capacitors, inductors, IC etc) to some of the signal lines.

Crosstalk

Crosstalk – Electrical Circuit

- Crosstalk are undesired coupling of signals from one circuit to another circuit.
- In a parallel bus context, the close placement of the data lines in PCB routing or cabling enables the effect of electrical signal in one trace/wire to be coupled over to the other. Creating undesired interference (crosstalk).
- Crosstalk can be transmitted electrically or via electromagnetic radiation (the trace/wire acts like an antenna).

Limitations signal skew & cross talk place on system design

>> max. no. of lines on data bus, max. clk rate, max. no. of bytes that can be transferred in one block, min. delay between each byte transfer

Parallel Data Transfer

Pros	Cons
>> Faster data transfer rate (more bits can be transfered at one time; given same clock rate as serial)	>> Signal Skew & Cross Talk (limits max clk speed & transfer distance)
>> Hardware interface design tend to be simpler as only strobe signals are needed.	>> Bulky hardware if data width is large
	>> need more space to route the PCB traces
	>> higher hardware costs

Serial Data Transfer

Pros	Cons
>> Less affected by Signal Skew & Crosstalk (less wires → able to support higher frequency data)	>> Data transfer rate lower given same clk rate since only 1 data line available
>> Transfer data reliably over a longer distance	>> more complex hardware interface as it needs to handle serial to parallel conversion (Processors typically only process in bytes, etc.)
>> Lower cost (less wires, etc.)	

Serial Data Transfer

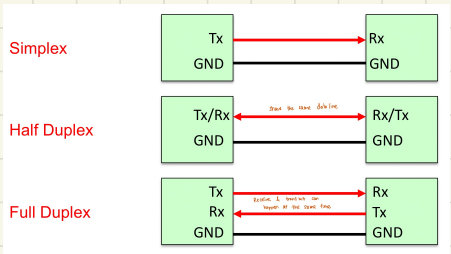
>> Data transferred one bit at a time over a single data line.

Serial Data Transfer Mode

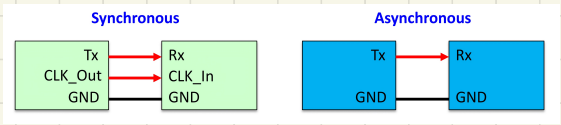
>> Simplex: Data transfer in 1 direction

>> Half-Duplex: Data transfer in both directions, but Rx & Tx can only be transmitter or receive at any one point in time

>> Full Duplex: Similtaneous Rx and Tx



Synchronous	Asynchronous
>> has common clock signal E.g. receive to latch in data at every rising edge of clock	>> no common clock signal; so devices have to agree on a pre-fixed clock frequency to use for data transfer



Synchronous Serial Transfer

>> Common clock signal between transmitter & receiver to synchronize the data transfer

>> Master - Slave Configuration

Calculations

MAX DATA TRANSFER RATE

>> Max strobe rate × No. of bits

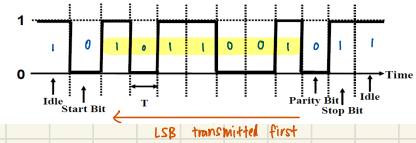
* Max strobe rate → determined by slowest device in the chain

E.g.

NOR Flash (NOR0001-1M)
- Max data strobe rate supported by DRAM = 10Mhz
- Max Processor Parallel bus strobe rate = 200Mhz
  ⇨ Max supported data strobe rate on this link = 10Mhz
- 16-bit interface => 16 bits data transferred with each strobe
  ⇨ 10M * 16 = 160Mbps.

DATA TRANSFER READING

2) **Fig. 2** shows the logical waveform of an asynchronous data frame.



>> 7-bit ASCII character transmitted.

revised in drawing ← 1001101_2 → 0x4D, 'M'

>> Even Parity: if Data + Parity bit = even no. of 1s

>> odd Parity: if Data + Parity bit = odd no. of 1s

>> if baud rate is 9600:

I cycle!

$T = \frac{1}{\text{baud rate}} = \frac{1}{9600} = 0.104$ ms

>> Data transfer rate of serial interface in characters per second (cps):

1 packet → 1 Start, 7 Data, 1 Parity & 1 Stop

Total: 10 bits

$\frac{\text{baud rate}}{\text{no. of bits per packet}} = \frac{9600}{10} = 960$ cps

>> if no parity bit: $\frac{9600}{9} = 1066$ cps

→ output is more but data packet integrity cannot be verified.

(f) Assume the baud rate of the transmitter is 4800, but the baud rate of the receiver is configured as 9600. Based on **Fig. 2**, determine, if any, the ASCII character(s) that will be received.
[Suggested Solution]

Possible multiple-sampling scenario

4800 baud rate: 7-bits: 1001101_2 → 0x4D

Original Frame								
S	D (lsb to msb)					P	E	
0	1011001					0	1	

Receiver Start bit to should be as well

Oversampled Frames (2x)										
Frame #1							Frame #2			
S	D (lsb to msb)					P	E	S	D (lsb to msb)	P	E
0	0110011					1	1	0	0001100	1	1

Frame #1: 0x66 → lowercase letter 'f'
Frame #2: 0x 18 → special character 'CAN'
Parity errors will be flagged for both frames (odd numbers of '1's received).

MAX BAUD RATE

Consider the system diagram in Figure 1 of case study notes.

- The processor UART peripheral in its Serial I/O module is configured to receive data with format of 1 start-bit, 7 data-bits, 1 parity-bit and 1 stop-bit.
- Each time the UART peripheral receives a character, it'll store the data into a buffer.
- It will then interrupt the CPU to notify CPU that there is data available.
- The CPU will then execute the Interrupt Service Routine (ISR) to read the character received.
- Interrupt Latency is the term used to describe the time between interrupt request and entrance to ISR.
- The minimum interrupt latency for the CPU is 10 µs and it takes 90 µs for the CPU to execute the instructions in the ISR (read the received data from the buffer).
- Assume that the UART data is transferred back to back i.e. no delays between each UART packets.

>> Total time required to transfer one character = Interrupt Latency + ISR execution

$T_c = \frac{1}{(10+90)}$ = 10000 ISRs per second

>> Each char transfer involved 10 bits on UART

10 bits * $\underbrace{10000}_{T_c}$ = 100 kbits per second

INTERRUPTS

The specifications of the system in Figure 1 (case study notes) requires the buttons to be asserted 400 times over a 24-hour period. Given that the processor consumes 0.1mA current when it is idling and 50mA when it is in active mode i.e. running code, answer the following.

(a) Supposed polled IO technique is used to sample the button status. Given that the processor poll the buttons every 100ms, each poll requires the processor to be active for 10ms, what is the average current consumed over the 24-hour period?

90 x 0.1 + 10 x 50 / 100

(a) Polled every 100ms, each poll takes 10ms
- ⇨ 90ms(idle), 10ms(active).
- ⇨ Average current = 0.9*0.1mA + 0.1*50mA = 5.09mA

(b) If interrupt-driven I/O is used to sample the button status, what is the average current consumed in the same 24-hour period? Given that it takes 20ms to service each interrupt, including interrupt latency and ISR execution.

(b) Interrupt mechanism.
- ⇨ Interrupted 400 times over 24-hour
- ⇨ Active time = 400*20ms = 8000ms
- ⇨ Average current consumed over 24-hour
  = [8*50mA + ((24*60*60)-8)*0.1mA] / [24*60*60]
  = 0.1046mA

Active : 400 times x 20ms = 8000 ms = 8 s

same: 8 x 50 + not active: (24x60x60 − 8) x 0.1

seconds in 24 hrs

(c) Given that a battery with capacity of 2000mAh means that it can supply 2000mA continuously for 1 hour, or 1000mA continuously for 2 hours. How long would a battery of 2000mAh last when used in (a) and (b) above?

(c) Use case (a) (Polling)
a. Average current consumed 5.09mA
b. Battery life = 2000mAh/5.09mA = 393 hours = 16.4 days

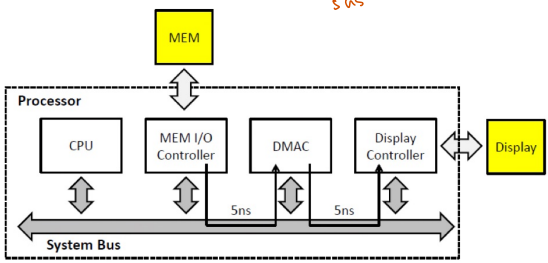
Use case (b) (Interrupt)
a. Average current consumed 0.104 mA
b. Battery life = 2000mAh/0.104mA = 19231 hours = 801 days = 2.2 years

battery capacity / average current

DMA

- The processor's system bus is capable of supporting simultaneous transfer of up to 3 bytes of data at one time.
- DMA is used to transfer video data from Memory to Display Controller module.
- Each video pixel data consist of three bytes (Red, Green, Blue) and are transferred simultaneously on the system bus on each bus cycle.
- Each transfer on the system bus takes 5ns and transfer of the bus control between the CPU and the DMAC takes 100ns.
- Assume that DMAC is using Fetch-and-Deposit DMA.
- Note that 1Kbyte = 1024 Byte.

5ns x 2



(a) Given that the video is output at a rate of 30 frames per second and each video frame has a resolution of 1920x1080 pixels. If burst-mode was used by the DMAC to burst one frame of video data at a time, would the DMAC be able to transfer each video frame completely?
[Suggested Solution]
Each transfer of the system bus will transfer 3 bytes of video data (RGB) and take 2*5ns.
Number of transfers in 1/30 seconds = 1920*1080 = 2073600 transfers.
Total time taken
= Bus control request + actual transfer + Bus release
= 100ns + 2073600*2*5ns + 100ns = 0.0207362 sec < 1/30sec
- ⇨ DMAC is able to transfer each frame of video before the start of the next frame request.

(b) Repeat the calculation if the DMAC is using cycle-stealing mode, assume that DMAC needs to wait at least 5 instructions before it could request for control of the system bus again.
[Suggested Solution]
Time taken for one transfer
= Bus control request + 1 transfer + Bus release + 5*CPU Clock cycles wait
= 100ns + 2*5ns + 100ns + 5*(1/400Mhz) = 222.5ns
Total time taken for 2073600 transfers = 2073600*222.5ns = 0.461376 sec > 1/30sec
- ⇨ DMAC will not be able to complete the transfer of one video frame before the next frame request comes in

Note that the last 5*CPU clock cycles wait is added in the above calculation to simplify the calculation process. But it'll not meet the requirement (>100%) even if it is taken into account.

Note: the '400 Mhz' for CPU clock is taken from the processor information in the case study notes.

Burst:
Bus control request + (no of frames x fetch + deposit) + Bus control request

Cycle Stealing:
Bus control request + (fetch + deposit) + Bus control release + instruction wait x 1/CPU clk

CAPACITY

(h) Part Number: HDD001

Overview	Electrical Interface
▪ 4 surfaces ▪ 1024 tracks per surface ▪ 128 sectors per track ▪ 512 bytes/sector ▪ Track-to-track seek time = 5 msec ▪ Rotational speed = 5000 RPM ▪ MTTF = 500,000 hours	• V(OH): 5V(max), 4.2V(min) • V(OL): 1.3V(max), 0V(min) • V(IH): 5V(max), 3.6V(min) • V(IL): 1.8V(max), 0V(min)

(i) Part Number: HDD002

Overview	Electrical Interface
▪ 8 heads ▪ 1024 cylinders ▪ 256 sectors per track ▪ 512 bytes/sector ▪ Track-to-track seek time = 4 msec ▪ Rotational speed =10000 RPM ▪ MTTF = 1,000,000 hours	• V(OH): 5V(max), 4.2V(min) • V(OL): 1.3V(max), 0V(min) • V(IH): 5V(max), 3.6V(min) • V(IL): 1.8V(max), 0V(min)

>> Formula for capacity:
Surfaces x tracks per surface x sectors per track x bytes/sector

>> Formula for access time:
T_A = Seek time + Average rotational delay

>> RPM → RPS : $\frac{5000\ rpm}{60s}$ → ~83 rps

a. What is the **capacity** of each drive?

[Suggested Solution]

HDD001:

Capacity = 4*1024*128*512 = 268, 435, 456 Bytes = 256Mbytes.
Note: 1Mbyte = 2^20 bytes

HDD002:

Capacity = 8*1024*256*512 = 1, 073, 741, 824 bytes = 1GByte.

Note:

1GByte = 2^30 bytes,

# of cylinders = # of tracks of each surface,

# of heads = # of valid surfaces

b. For HDD001,
   i. What is its **access time**?

[Suggested Solution]

RPS = 5000/60

Average rotational delay = $T_{R.AV}$ = 0.5/RPS = 0.5/(5000/60) = 6ms

Seek time = T_S = 5ms

Access time, $T_A = T_S + T_{R.AV}$ = 5 + 6 = 11ms

ii. What is the time needed to transfer a 4Kbyte file stored in random non-consecutive sectors on different tracks? Assume that every sector is on a different track.

[Suggested solution]
RPS = 5000/60 per second
Access time T_A = 11 ms

# of sectors = 4*1024/512 = 8
Transfer time T_T for 1 sector = 512/(RPS*D_T*D_S) = 512/((5000/60)*128*512)
= 93.75 us
Total time, T_{TOTAL} = 8*(T_A + T_T) = 88.75 ms

Note: This illustrate the effect for HDD fragmentation. If a file is fragmented, i.e. stored in non-consecutive sectors/clusters, the effective performance of the HDD data transfer will drop, as shown by the total access time for the two examples above.

iii. After defragmenting HDD001, what would be the time needed to transfer a 280Kbyte file?

[Suggested solution]
Defragmentation => file is stored in consecutive sectors.

However, 280KByte is large than size of one track (128*512=64KByte).

It occupies 4 full tracks and 24 Kbyte on the fifth track.

RPS = 5000/60 per second
Access time T_A = 11 ms

T_T for one full track = 1/RPS = 12 ms

Total time for 1 track, T_{TOTAL} = 11+12 ms = 23 ms

4 tracks = 4*23 = 92ms

Last 24KByte:
Transfer time T_T = N/(RPS*DT*DS) = (24*1024)/((5000/60)*128*512) = 4.5 ms
Total time, T_{TOTAL} = 11+4.5ms = 15.5 ms

Total time needed to transfer 280KByte file = 92+15.5 = 107.5 ms

Note: T_A is required when moving from one track to the other since the R/W head need to be positioned at the starting sector of the next track.

TUTORIAL #8	Cache and Virtual Memory	(Cx1106)
	Solutions	

Note: <u>Do not</u> refer to the processor configuration in the case study notes for this tutorial. A smaller system will be used instead.

8.1 Cache

1. Given a processor system with the following characteristics
   - Processor has a direct-mapped cache with 32 cache blocks and a cache size of 512 bytes.
   - Cache Memory Access time = 5ns.
   - Cache Hit rate = 0.9
   - 64Kbyte DRAM used as the main memory.
   - DRAM Memory access time = 200ns

   a. In doing cache mapping analysis, how many **blocks** would the main memory be partitioned to?

   [Suggested Solution]

   cache block size: cache size / cache blocks

   Cache block size = 512/32 = 16 Bytes

   Number of main memory blocks = 64KByte/16Byte = (2^16)/(2^4) = 2^12 = 4096

   b. What is the format of a memory address as seen by the cache (i.e. determine the sizes of the tag, block and offset fields)?

   [Suggested Solution]

   Cache Block Size = 16 => Offset Field = 4 bits

   Number of blocks in the cache = 32 => Block Field = 5 bits

   Main Memory Size = 64KByte => 16 address bits

   Tag Field bits = Main Memory Address bits – Offset – Block = 7

   ⇨ TAG:BLOCK:OFFSET = 7:5:4

   c. CPU needs to read a byte from main memory address 0x0DB63.

      i. Which cache block would CPU looked at to search for the required data?
      ii. How many main memory blocks could potentially be mapped to the same cache block as that of 0x0DB63?
      iii. How does the CPU knows if the cache block identified in (i) above contains the data that it needs?
      iv. What is the purpose of the 'offset' field in the cache mapping?

[Suggested Solution]

i. 0x0DB63 = **0000 1101 1011 0110 0011**
⇨ Block 10110b = Block 22
ii. Number of MM blocks = 4096
Number of cache blocks = 32
=> each cache block is a potential destination to 4096/32 = 128 MM blocks.
iii. By looking at the TAG value entry of the corresponding cache block
For 0x0DB63, the TAG value for the cache block should be equal to **00001101 101b**
iv. Offset refers to the offset of the byte of interest from the cache block boundary. For 0x0DB63, the byte is the byte 3 of block 22. First byte of the block is byte 0.

d. What is the effective access time of the memory in this system?

[Suggested Solution]

Assume cache memory and main memory access do not overlap.

EAT = H*$Cache_{Access}$ + (1-H)*($Cache_{Access}$ + Mem_{Access}) = 0.9*5ns + (1-0.9)*(5ns+200ns) = 25ns.

8.2 Virtual Memory

2. In a processor system with the following characteristics,
   - 1 MByte Virtual memory space
   - 64 Kbyte DRAM as main memory
   - Paging scheme used for virtual memory management, Page Table as shown in Table 8.2
   - Virtual Page size = 1 KByte
   - TLB with 4 entries

Table 8.2 – Page Table

Virtual Page Number	Valid Bit	Page Frame Number
0	1	1
1	1	2
2	0	-
3	1	16
4	1	9

1 byte = 8 bits

a. How many bits are required for each virtual address?

[Suggested Solution]

1Mbyte = 2^20 byte => 20 bits

1 Kb = 2^{10}

1 Mb = 2^{20}

1 Gb = 2^{30}

b. How many bits are required for each physical address?

[Suggested Solution]

64 KByte = 2^6 * 2^10 = 2^16 => 16 bits

c. What is the maximum number of entries in the page table in Table 7.2?

[Suggested Solution]

Number of virtual pages = 2^20 / 2^10 = 2^10 = 1024

⇨ Max number of entries = 1024

d. What is the maximum number of valid entries in the page table in Table 7.2?

[Suggested Solution]

Number of Physical Frames = 2^16 / 2^10 = 64

⇨ Max number of valid entries= 64

e. With reference to Table 7.2, answer the following. Indicate when a page fault occurs.

(i) The compiler mapped the UART routine to virtual address 0x005F0 – 0x006FF, where in the DRAM would you be able to find the UART routine?

[Suggested Solution]

0x005F0 ➔ virtual page number 1 ➔ mapped to page frame number 2
0x006FF ➔ virtual page number 1 ➔ mapped to page frame number 2

Virtual Address (20 bits)	Physical Address (16 bits)
0000 0000 0101 1111 0000	0000 1001 1111 0000
0000 0000 0110 1111 FFFF	0000 1010 1111 FFFF

Physical Address = 0x09F0 – 0x0AFF

(ii) The compiler mapped the I2C routine to virtual address 0x009C0 - 0x009DF, where in the DRAM would you be able to find the I2C routine?

[Suggested solution]

0x009C0 ➔ virtual page number 2 ➔ page fault occurs (valid bit =0).

0x009DF ➔ virtual page number 2 ➔ page fault occurs (valid bit =0).

(iii)What happens when there is a page fault?

[Suggested Solution]

Page fault => the required data/code is not in the main memory

=> OS needs to retrieve the required information from the storage memory and update the paging table accordingly.

f. What memory are the Page Table and TLB resided?

[Suggested Solution]

Main Page Table => Main Memory

TLB => Internal Fast Memory close to CPU

g. What is the function and effect of a TLB?

[Suggested Solution]

⇨ Cache the frequently used Page table information
⇨ Leads to shorter access time for paging information => increase in system performance.

(Not necessary to be covered during tutorial)

[Optional, but students are encouraged to attempt these questions]

3. Consider a system with the following characteristics.
   - Direct mapped cache of 32 cache blocks and cache block size of 32 bytes
   - Cache uses Physical Address for address mapping
   - Virtual Memory page size 2048 bytes
   - Virtual Memory size is 1Mbyte. Physical Memory size is 64KByte
   - Extracts of Page Table (valid entries)
     - Virtual Page 0 → Physical Frame 9
     - Virtual Page 1 → Physical Frame 3
     - Virtual Page 2 → Physical Frame 5
     - Virtual Page 3 → Physical Frame 2
     - Virtual Page 4 → Physical Frame 7
   - The main program is 5KByte in size and starts at virtual address 0x01006

   (i) Assuming that the compiler allocates the program sequentially in the virtual memory, what is the physical address of the start and end of the main program?

[Suggested Solution]

Virtual Address (Start) = 0x01006, end = 0x02406. (5KByte size)
0x1006 => 0001 0000 0000 0110 => Page 2 => Physical Frame 5 (101b)
Physical address = 0010 1000 0000 0110 = **0x2806**
0x2406 => 0010 0100 0000 0110 => Page 4 => Physical Frame 7 (111b)
Physical Address = 0011 1100 0000 0110 => **0x3C06**

   (ii) Which cache block should the CPU check in the cache for the start of the main program? What is the corresponding TAG value used to check for cache hit/miss?

[Suggested Solution]

Physical Address used for tagging. Hence, TAG:BLK:OFFSET = 6:5:5
Start address = 0x2806 = 0010 1000 0000 0110
TAG value = 001010 = **0xA**, BLK = 000000 = **0**

TUTORIAL #9	Computer Arithmetic and Processor Pipeline	(Cx1106)

Solutions

9.1 Computer Arithmetic

1. Describe and explain the pros and cons of fixed and floating number system, particularly the representable range and precision. Illustrate using example of 32-bit fixed point and the 32-bit floating point number in single precision IEEE754 format.

[Suggested Solution]

- Floating Point (IEEE754)
  - Max Range ≈ $2*2^{128}$ (~2^{128} to -2^{128}).
  - Max Precision (near to zero) less than 2^{-126}
- Fixed Point
  - Max Range (Radix right of LSB, unsigned) ≈ 2^{32}
  - Max Precision (Radix left of MSB) 2^{-32}
- Floating point yield a larger range and better precision at small numbers with the same number of bits representation.
- One usually needs the best precision when the numbers are small.
- However, Fixed point number has the advantage of having uniform precision across entire range.
- Floating point number's precision changes across the range and the very coarse precision at the two end of the range may not be desirable to the intended algorithm.

2. An array consisting of the length of 256 wires is given by L[0], L[1], … , L[255]. Assume that user are not allowed to test for any overflow during computation, describe a scheme to compute the average length of the 256 wires that will yield a result with the highest precision based on the following specifications:

   - 16-bit registers are used for storing the data and result.
   - Only Single-Precision (16-bit) and Fixed-Point arithmetic is used.
   - Maximum possible length of each wire is 0x3FF and is an integer.

   No coding is required, illustrate your answer in the form of a mathematical expression and justify your answer.

[Suggested Solution]

- Unsigned arithmetic used to maximise the range since the wire length cannot be negative
- 16-bit => maximum 0xFFFF
- Max length of one wire = 0x3FF => Max number of wire length that can be accumulated before the result will potentially overflow = 64
- Math Expression =
  {(L[0]+…+L[63])/64 + (L[64]+…+L[127])/64 +
  (L[128]+…+L[191])/64 + (L[192]+…+L[255])/64}/4

Always perform Division last to avoid truncating the LSBs too early in the computation. But need to take note of overflow when using addition, subtraction and multiplication.

9.2 Pipelines

3. Consider a processor (not ARM processor) with 4 pipeline stages: Fetch Instruction (F), Decode (D), Execute (E) and Store (S). Assume that

   - Branch target address is calculated at the execute stage
   - Instruction length for every instruction is one word long
   - Each pipeline stage take 1 cycle to complete
   - Delay Branching is not enabled.

   How many cycles does the code in Figure 9.2 take? You can ignore all pipeline conflicts you see in the code, just focus on the pipeline behaviour and execution cycles.

```
      MOV   R6, #0x900 ; I1
      MOV   R5, #0     ; I2
      MOV   R4, #0x800 ; I3
      MOV   R3, #0x300 ; I4
Loop  LDR   R0, [R3]   ; I5
      LDR   R1, [R4]   ; I6
      ADD   R2, R1, R0 ; I7
      ADD   R5, R5, #1 ; I8
      ADD   R4, R4, #1 ; I9
      ADD   R3, R3, #1 ; I10
      CMP   R5, #5     ; I11
      BNE   Loop       ; I12
      ADD   R4, R4, R2 ; I13
      STR   R2, [R6]   ; I14
```

Figure 9.2

[Suggested Solution]

- I1 to I4: 7 cycles (I1 will take 4 cycles to pass through the pipeline)
- I5 to I12. For the first four iterations, each iteration takes (8+2) cycles. This includes 2 cycles discarded in each iteration.
- I5 to I12. Fifth iteration takes 8 cycles since the branch is not taken so no instructions are discarded.
- I13 to I14: 2 cycles.
- Total = 7 + (8+2)*4 + 8 + 2 = 57 cycles.

4. Consider a processor (not ARM processor) with 4 pipeline stages: Fetch Instruction (F), Decode (D), Execute (E) and Store (S). Assume that

   - Branch target address is calculated at the execute stage
   - Instruction length for every instruction is one word long
   - Each pipeline stage takes 1 cycle to complete
   - No Resource Conflicts
   - Delayed Branching is enabled

   Identify and describe ALL pipeline conflicts the code in Figure 9.3 has when executed in the pipeline processor above. Suggest workaround for pipeline conflicts identified.

```
      MOV   R6, #0x900 ; I1
      MOV   R5, #0     ; I2
      MOV   R4, #0x800 ; I3
      MOV   R3, #0x300 ; I4
Loop  LDR   R0, [R3]   ; I5
      LDR   R1, [R4]   ; I6
      ADD   R2, R1, R0 ; I7
      ADD   R5, R5, #1 ; I8
      CMP   R5, #5     ; I9
      BNE   Loop       ; I10
      ADD   R4, R4, #1 ; I11
      ADD   R3, R3, #1 ; I12
      ADD   R4, R4, R2 ; I13
      STR   R2, [R6]   ; I14
```

Figure 9.3

[Suggested Solution]

- I4 and I5. Data dependency. R3 register value is used in I5 before I4 completes loading R3 register. Resolved by
  - swapping I3 and I4, OR swapping I5 ad I6, OR
  - inserting a NOP instruction between I4 and I5
- I6 and I7. Data dependency. R1 register value is used in I7 before I6 completes loading R1 register. Resolved by
  - swapping I7 and I8, OR
  - inserting a NOP instruction between I6 and I7
- I8 and I9. Data dependency. R5 register value is used in I9 before I8 completes loading R1 register. Resolved by
  - swapping I7 and I8, OR
  - inserting a NOP instruction between I8 and I9

- I5 and I12. Data dependency. I11 and I12 are delay slot instructions. During the first four iterations of the loop, I5 uses R3 before I12 can load the latest increment to R3. Resolved by
  - swapping I11 and I12, and add a NOP before I13.
- No branch conflicts since delayed branching is used.
- No data dependency between I13 and I14 since I14 is just reading and not writing to R2

(Not necessary to be covered during tutorial)
[Optional, but students are encouraged to attempt these questions]

9.3 Rounding Error

5. You have been tasked to write a program that calculates the actual time based on a counter that is incremented once every 0.10 seconds. For example, if the counter value is 3,600,000, you would expect the actual time to be 100 hours ((3,600,000 x 0.1) / (60 x 60)).

   Suppose you have decided to use a 24-bit fixed point representation as shown in Figure 8.4 to store the value of 0.10 seconds ($2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + \ldots$).

0	▪	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0

Figure 9.3 – Fixed point representation of 0.1010

   a. Approximate the round-off error (in decimal) of 0.10_{10} due to the fixed-point representation.
   b. What is the effect of this round-off error on the time calculated if the counter value is 3,600,000?

[Suggested Solution]

a. The decimal value of the fixed point representation in Figure 8.4 is 0.0999999046325684_{10}.

   Hence, the round-off error is approximately **0.000000095_{10}** (0.10_{10} - 0.0999999046325684_{10}).

b. The time calculated will be off by approximately **0.34 seconds** (3600000_{10} x 0.000000095_{10}).

Note: This question is based on an actual example of disasters caused by bad numeric. The inaccurate calculation of time has contributed to fatalities when it was used for intercepting enemy missiles during Gulf War. While an error of 0.34 seconds may not seem like much, a missile travelling at 1,676 meters per second would have travelled approximately 500 meters in that time.

Friend's Notes

Important Formulas (1106)

HDD Transfer Rate:

Revolutions Per Second (RPS) = RPM/60

Average Rotational Delay ($T_{R,Av}$) = $\frac{0.5}{RPS}$

Transfer Time (T_T) = $\frac{N}{RPS * D_T * D_S}$ → N: size of file

- RPS: Revolutions per sec
- D_T: Track Density
- D_S: Sector Density

Access Time (T_A) = $T_S + T_R$

- T_S: Average Seek time
- T_R: Average Rotational Delay

Direct-Mapped Caching

Main/System Memory Size = 64kBytes

Cache Size = 256 Bytes

Cache Block Size = 16 Bytes

Consider a system with the following characteristics.

- Direct mapped cache of 16 cache blocks and block size 16 bytes
- Cache uses Virtual Address for address mapping
- Virtual Memory page size 1024 bytes
- Virtual Memory size 1Mbyte. Physical Memory size 64 KByte
- Extracts of Page Table (valid entries)
  - Virtual Page 0 → Physical Frame 2
  - Virtual Page 1 → Physical Frame 5
  - Virtual Page 2 → Physical Frame 8
  - Virtual Page 9 → Physical Frame 3

From which cache block would the CPU retrieve the data for a virtual address 0x00119? What is the corresponding tag value of the cache block if it is a cache hit?

(5 marks)

Where would Data at main memory address 0X1106 be mapped to?

Offset bits = $\log_2$(Cache Block Size) = $\log_2(16) = 4$

No of Cache blocks = 256/16 = 16

Blk bits = $\log_2$(Number of Cache Blocks) = $\log_2(16) = 4$

Main Memory Size = 64kB (Convert to bytes) = 64 * 1024 = 65536

Main Memory Size in bits = $\log_2(65536) = 16$ bits

Tag bits = 16 − 4 − 4 = 8

Mapping Format = 8:4:4

Apply to the main memory address 0X1106 (convert to binary)

0X1106 = 0001 0001 0000 0110b

- 0001 0001: # TAG
- 0000: #Blk
- 0110b: # Offset

TAG = 0X11 Blk = 0X0 Offset = 0X6

$\log_2(16) = 4$

$\log_2(16 = 4$

$64 * 1024 = 65536$

$\log_2(65536) = 16$

$16 - 4 - 4 = 8$

TAG

8 4 4

0001 0001 0000 0110

tag Blk offset

Cache Miss Rate → 1 − Hit Rate → (1 − H) ẅ

Effective Access Time

$$EAT = H \times Access_c + (1 - H) \times (Access_c + Access_{MM})$$

- H: Hit Rate
- $Access_c$: Access time for cache
- $(1 - H)$: Cache Miss Rate
- $Access_c$: Access time for cache
- $Access_{MM}$: Access time for main memory

Example: Consider a system with a main memory access time of 200ns supported by a cache having 10ns access time and a hit rate of 99%.

Hit Rate = 0.99

$Access_c$ = 10ns

$Access_{MM}$ = 200ns

$0.99 \times (10) + (0.01)(10 + 200)$

$= 12ns$

Paging Example:

Virtual Address Space → 8 kb → 8192 bytes → 2^{13} → 13 bits

Physical Address Space → 4kb → 4096 bytes → 2^{12} → 12 bits

Page Size → 1 kb → 1024 bytes → 2^{10} → 10 bits (offset)

Virtual Page field = 13 - 10 = 3

Physical Frame field = 12 - 10 = 2

Virtual Address Examples:

0X1553 / 0X0FA0

1. 0X1553 (Break it up to binary)

Follow the virtual addr Space 13 bits, therefore for 1, omit the 0s

1 0101 0101 0011 13 bits representation

virtual page field — offset

101 in decimal is 5. Check page 5 if there is a valid bit. It is found in frame 1.

In physical frame → 0101 0101 0011

physical frame — offset retained

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

2. 0X0FA0 (Break it up to binary)

0 1111 1010 0000 13 bits representation

virtual page field — offset

011 in decimal is 3. Check the Page 3 if there is a valid bit. It has a valid bit of 0, therefore, there is a page fault.

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

Arithmetic

Two's Complement Representation

MSB = 0 (positive number)

MSB = 1 (negative number)

Example: (Convert to Negative)

+106: 0 1 1 0 1 0 1 0

-106: 1 0 0 1 0 1 1 0

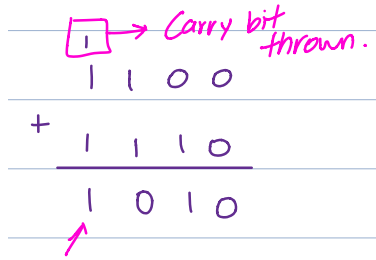
(After the first 1, negate the rest)

$-128+16+4+2 = -106$ ✱

Examples of Carry and Overflow:

(-4): 1100

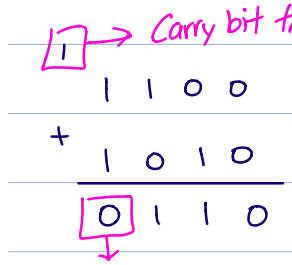
(-2): 1110



No overflow, sign still same

(-4): 1100

(-6): 1010

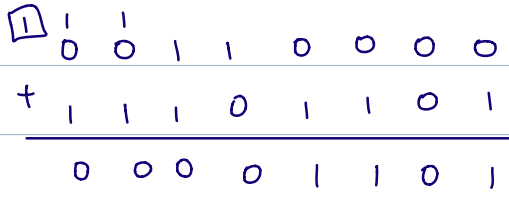


Sign change, overflow occured

Carry Bit

Example: (48+(-19) = 29)

Carry bit thrown away ~



Overflow (For signed numbers)

In (A+B): (Addition)

MSB(A) = MSB(B), MSB(Result) ≠ MSB(A)

Overflow occured

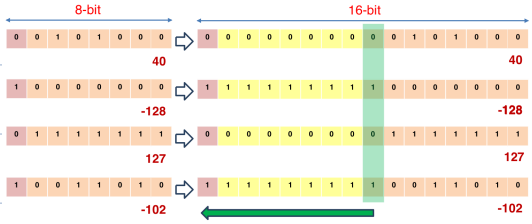
In (A-B): (Subtraction)

MSB(A) ≠ MSB(B), MSB(Result) ≠ MSB(A)

Overflow Occured

Sign Extension

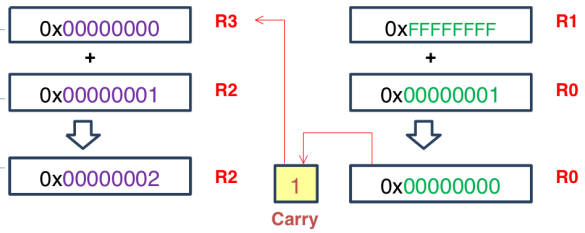
- In two's complement, sign extension is needed to convert a smaller size operand to a larger size operand



- Sign extension simply copies the sign bit (MSB) into the higher order bits

Multi-Precision Arithmetic

```
ADD   R0,R0,R1 ; add lower word with carry out
ADC   R2,R2,R3 ; add upper word with carry in
```



Range and Precision:

Floating Point Representation: $\pm$ 9.99 e $\pm$ 99

Sign, Mantissa, Exponent

Floating Point can be represented from (-9.99×10^{99} to $+9.99 \times 10^{99}$)

Normalisation: Avoid Synonymous Representation by maintaining one non-zero digit before radix → $\pm$ 5.42 e $\pm$ 03 (Normalised)

Non-zero

Underflow: (occurs when value is too small to be represented)

Smallest positive Normalised number: + 1.00 e -99

Smallest negative Normalised number: - 1.00 e -99

Floating Point Standards

Single Precision (32 bits)

1 bit Sign + 8 bit Exponent + 23 bit Fraction

Double Precision (64 bits)

1 bit Sign + 11 bit Exponent + 52 bit Fraction

Conversion of Single Precision to decimal

$(-1)^S \times (1.F)_2 \times 2^{E-Bias}$

Sign bit → S=0 (positive), S=1 (negative)

Exponent → Binary number left on radix

Value of Exponent = E - Bias (127 for Single, 1023 for Double)

Fraction → Example → $(1.11)_2$ → $(1 + 1\times2^{-1} + 1\times2^{-2})$ (Expressed this way)

Example: 0 1 0 1 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sign = 0

Fraction = $(1.111)_2 = 1 + 2^{-1} + 2^{-2} + 2^{-3} = 1.875$

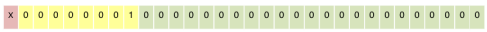
Exponent = 1011 0010 = 178

Value in decimal = $+1.875 \times 2^{51}$

E - Bias = 178 - 127 = 51

Representable Range for Normalised Single Precision

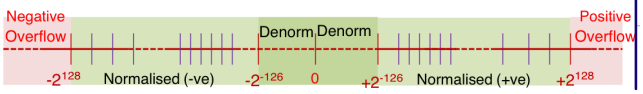
- In normalised mode, exponent is from 00000001 to 11111110
- Smallest magnitude normalised number



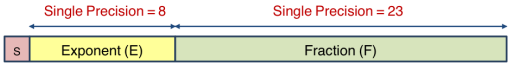
Exponent = $(00000001)_2$ = 1; E - Bias = 1 – 127 = -126
1 + Fraction = $(1.000\ldots000)_2$ = 1
Value in decimal = 1×2^{-126}

- Largest magnitude normalised number

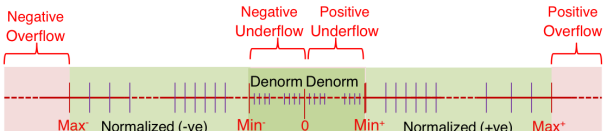
Exponent = $(11111110)_2$ = 254; E - Bias = 254 – 127 = 127
1 + Fraction = $(1.111\ldots111)_2 \approx 2$
Value in decimal = $2 \times 2^{127} \approx 2^{128}$



IEEE 754 Encoding



Mode	Sign	Exponent	Fraction
Normalized	1 / 0	00000001 to 11111110	Anything
Denormalized	1 / 0	00000000	Non zero
Zero	1 / 0	00000000	0000 … 0000
Infinity	1 / 0	11111111	0000 … 0000



Addition of floating point numbers

Rounding Error Mitigation

Suppose Given : $1.23 e + 01$ (steps to take)

$4.56 e + 00$

1. Add Guard digit to maintain precision

$1.230 e + 01$

$4.560 e + 00$

2. Align the digits

$1.230e+01$
$0.456e+01$
} $1.686 e+01$

3. Round Off → $1.686 e + 01$ → $1.69 e+01$

Example:

$1.23 \times 10^3 + 1.00 \times 10^0 +$
$1.00 \times 10^0 + 1.00 \times 10^0 +$
$1.00 \times 10^0 + 1.00 \times 10^0$

Align the 1.00×10^0 Exponent

1 $1.230\ e\ +03$

2 $0.001\ e\ +03$ (Align)

3 $1.231\ e\ +03$ (Round off)

4 $1.230\ e\ +03$ (Repeat of 1,2,3)

$+\ 0.001\ e\ +03$

⋮

$1230\ e\ +03$ Final Ans

Example:

$1.00 \times 10^0 + 1.00 \times 10^0 +$
$1.00 \times 10^0 + 1.00 \times 10^0 +$
$1.00 \times 10^0 + 1.23 \times 10^3$

$+\ 1.000\ e\ +00$
$1.000\ e\ +00$
$2.000\ e\ +00$
⋮
$5.000e\ +00$ (needs Alignment)
$+\ 1.230e\ +03$
↓
$0.005c\ +03$ (Aligned)
$+\ 1.230c\ +03$
$1.235\ +03$ (Round)
$1.240\ +03$ (Rounded value)