FIT9137 Applied Session

Week 3

Topics:

- The purpose of this applied session is getting to know memory design and management.

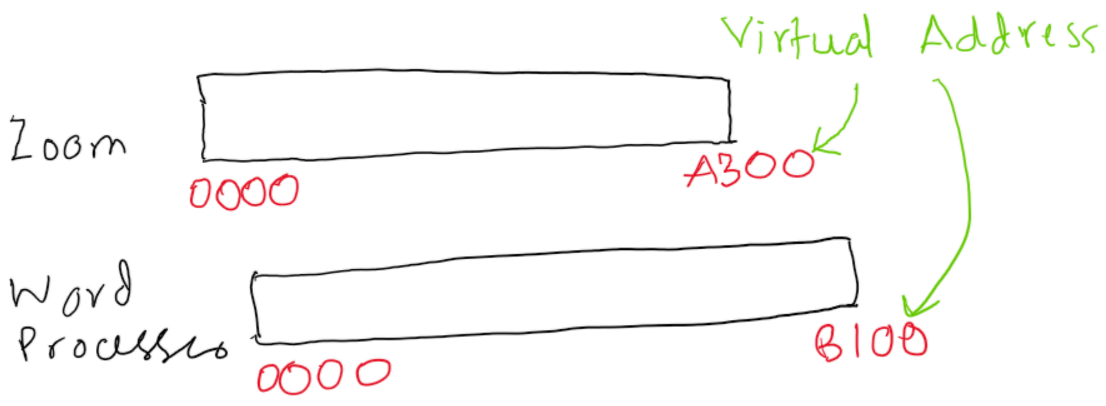
Instructions:

- One of the main purposes of this applied session is to understand memory addressing techniques extending the knowledge gained in the previous sessions using MARIE simulator. The other goal is to give and receive feedback from your peers and or your tutors.
- Form groups of 2 students (peers) to work through the exercises. If you meet a problem, try to solve it by asking direct questions to your peers. If the issue was not solved within peers, ask your tutor. If you did not get a chance to solve the problem during your applied session with your peer or tutor, jump into one of many consultation hours and ask any of the tutors to help you. Please visit the "Teaching Team and Unit Resources" tile in the FIT9137 Moodle site.

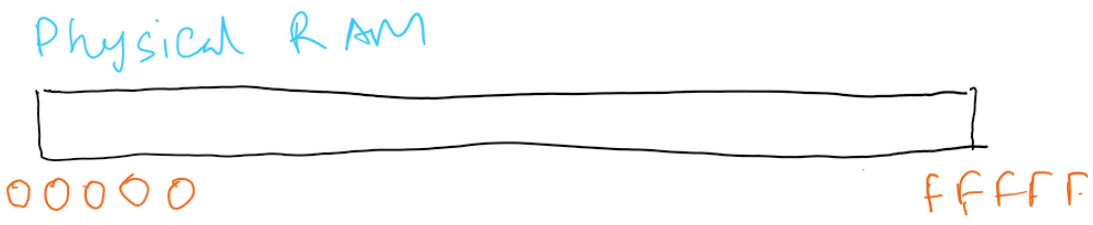
1. What are the different types of memory present in a computer? Explore your own laptop memory. Can you present a schematic drawing showing each of them?

2. Virtual memory addresses map to physical memory addresses

Assume we have two processes: a zoom application and a word processor running in a computer. The virtual addresses for these processes are given below:



To demonstrate the mapping from virtual to real memory addresses, we assume the real memory addresses available for a computer are as below.



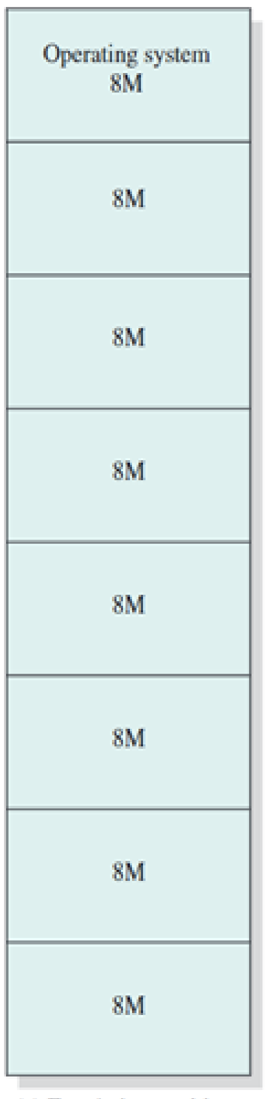
The Operating system allocates the following beginning addresses for these two processes.
(i) the zoom application is stored beginning at memory location 10000_H and
(ii) the word processor is stored beginning at memory location 40000_H

After these programs are brought into the memory, what should it look like? Draw the layout of the physical memory and find the real memory address of the end address of the above processes.

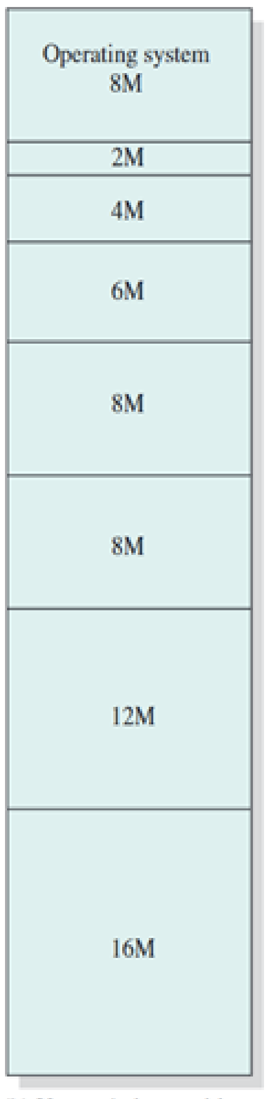
3. Memory allocation using Fixed Partitioning (equal size and unequal size)

There are three processes of the following sizes. Allocate these processes to RAM (physical memory) on the two systems below (System A and System B). Then, calculate the internal fragmentation for each allocation.

- Process A – 512KB
- Process B – 4MB
- Process C – 10MB



(a) Equal-size partitions



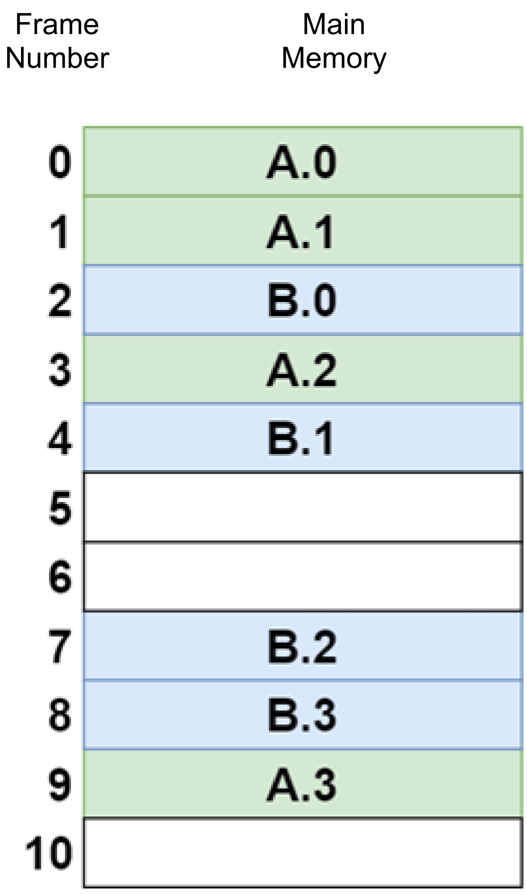
(b) Unequal-size partitions

4. Memory Allocation using Dynamic Partitioning

A computer system uses dynamic memory partitioning and currently holds three processes of sizes 20MB, 18MB and 14MB with 4MB of free space available. Draw the memory diagram showing the processes. Show the processes in the memory after each of the following events (i) the 14MB process completes its execution (ii) a new process of size 8M is brough in, (iii) 20MB process completes its execution, (iv) a new process of size 14MB is brough in, (v) What's the total available free space in the RAM (vi) can the OS bring in a new process of size 12MB? (vii)If not, how can the OS make space for the new process without moving any other processes out of the memory.

5. Memory Allocation using Paging

Following is a diagram of the frames on a physical memory of a system where paging is used for memory allocation. The Page Size and Frame Size of this system is 4KB



1. Can you create the Page tables for Process A and Process B?
2. If Process A is 15KB and Process B is 13KB, what's the Internal Fragmentation caused when allocating these 2 processes to the above system.
3. There is a new Process C of size 10KB which needs to be allocated to the memory. If the CPU allocates the pages starting from the first available free frame in the RAM, create the Page Table for Process C when it gets placed in the RAM.
4. There is another new Process D of size 11KB which needs to be allocated to the memory. The CPU decides to move Process B, which was an idle process from the RAM to the Disk to make space for the new Process D in the RAM. Create the Page table for Process D.
5. The user now wants to access Process B again. Discuss what happens next and how the CPU is going to accommodate this request.

Additional Tasks (Optional)

1. Indirect (Memory) Addressing in MARIE

In MARIE, we can make use of the instructions *"LoadI"* and "*StoreI*" (handling indirect addressing) to store and retrieve data from MARIE memory.

```
          ………………..
          StoreI  MemAdd
          LoadI   MemAdd
          ………………….
MemAdd,           HEX 020
```

Referring to the example code above, *StoreI* command will store the content of AC into the memory location "HEX 020," and *LoadI* command will load the content of the memory location "HEX 020" into the AC.

Write a MARIE program to input decimal numbers from the keyboard and store them starting from memory location "HEX 020." The program should terminate when the user enters a number '0'. Note: one decimal number (e.g. 20) to be stored in one memory location.

2. Handling ASCII Characters in MARIE

In MARIE assembly language programming, we can make use of the ADR command, the HEX keyword and a label "myName" to store this string in memory:

```
myAdd,      ADR    myName
myName,     HEX 059      /'Y'
            HEX 06F      /'o'
            HEX 075      /'u'
            HEX 072      /'r'
            HEX 02D      /'-'
            HEX 04E      /'N'
            HEX 061      /'a'
            HEX 06D      /'m'
            HEX 065      /'e'
            HEX 0        /NULL
```

Here, the label "myAdd" refers to the memory location of the label "myName". So, "myAdd=0001" refers to the first character 'Y.'

Instr. No.	Memory Location	Label	Code	Memory Contents
1	000	myAdd,	ADR myName	0001

2	001	myName,	HEX 059 /'Y'	0059
	002		HEX 06F /'o'	006F
	003		HEX 075 /'u'	0075
	004		HEX 072 /'r'	0072
	005		HEX 04E /'N'	004E
	006		HEX 061 /'a'	0061
	007		HEX 06D /'m'	006D
	008		HEX 065 /'e'	0065
	009		HEX 0 /NULL	0000
	00A			

Figure 1

Write a MARIE program to load the characters (into the AC) one-by-one using indirect addressing (LoadI) and modify the character by manipulating the ASCII code (adding '1' to the ASCII value) and store the character back into the same memory location using StoreI command. Verify the changed ASCII values in the MARIE memory.