**W2.4:**

For e.g. consider an 8 bit unsigned integer with a scaling factor of 1/8, then:

There are three types of floating-point numbers:

subnormal numbers (the exponent is 0x00) which use the formula:

normalized numbers (the exponent is between 0x01 and 0xFE) which use the formula:

special numbers (the exponent is 0xFF) if m=0 we have +- infinity, otherwise we have NaN.

IEEE754 floating-point numbers have an implicit leading bit in the significant with normalized numbers.

**W4.7:**

#include <sys/types.h>

#include <sys/stat.h>

 #include <unistd.h>

int open(const char \*pathname, int flags);

int open(const char \*pathname, int flags, mode\_t mode);

int close(int fd);

ssize\_t read(int fd, void \*buf, size\_t count);

ssize\_t write(int fd, const void \*buf, size\_t count);

int truncate(const char \*path, off\_t length);

int ftruncate(int fd, off\_t length);

off\_t lseek(int fd, off\_t offset, int whence);

int ioctl(int d, int request, ...);

int stat(const char \*path, struct stat \*buf);

int fstat(int fd, struct stat \*buf);

int lstat(const char \*path, struct stat \*buf);

c provides two similar approaches for reading/writing files. These are:

**System Calls** - set of libraries that gives you direct assess to the underlining system calls. This doesn't provide much to the programmer, although, it gives the programer complete control of this resource. Most of these functions are in volumn 2 of the unix man pages. They include: open, read, write, ioctl, ...

**File streams** - this provides a slightly higher level abstraction on files providing some buffering of the data. Most of these functions are in volume 3 of the unix man pages. They include: fopen, fread, fwrite, ...

**W9 interrupt**

The interrupt handler routine is similar to a normal routine, however, it will also normally return the registers to an identical state to what they were when the interrupt occurred and the routine will involve the unmasking of interrupts.

"Interrupt - An event that alters (or interrupts) the normal fetch-decode-execute cycle of execution in the system." Glossary of Computer Organization and Architecture Ed. 2, Null and Lobur. When an interrupt occurs the currently executing program is suspended and the program counter jumps to the interrupt handler (normally at a fixed location in memory). Once the interrupt handler completes the suspended program is restarted. The interrupt handler must return the CPUin exactly the same way it finds it. Hence the interrupted program is 'unaware' that the interrupt has every taken place.

0x0001 : jump iodevhandler

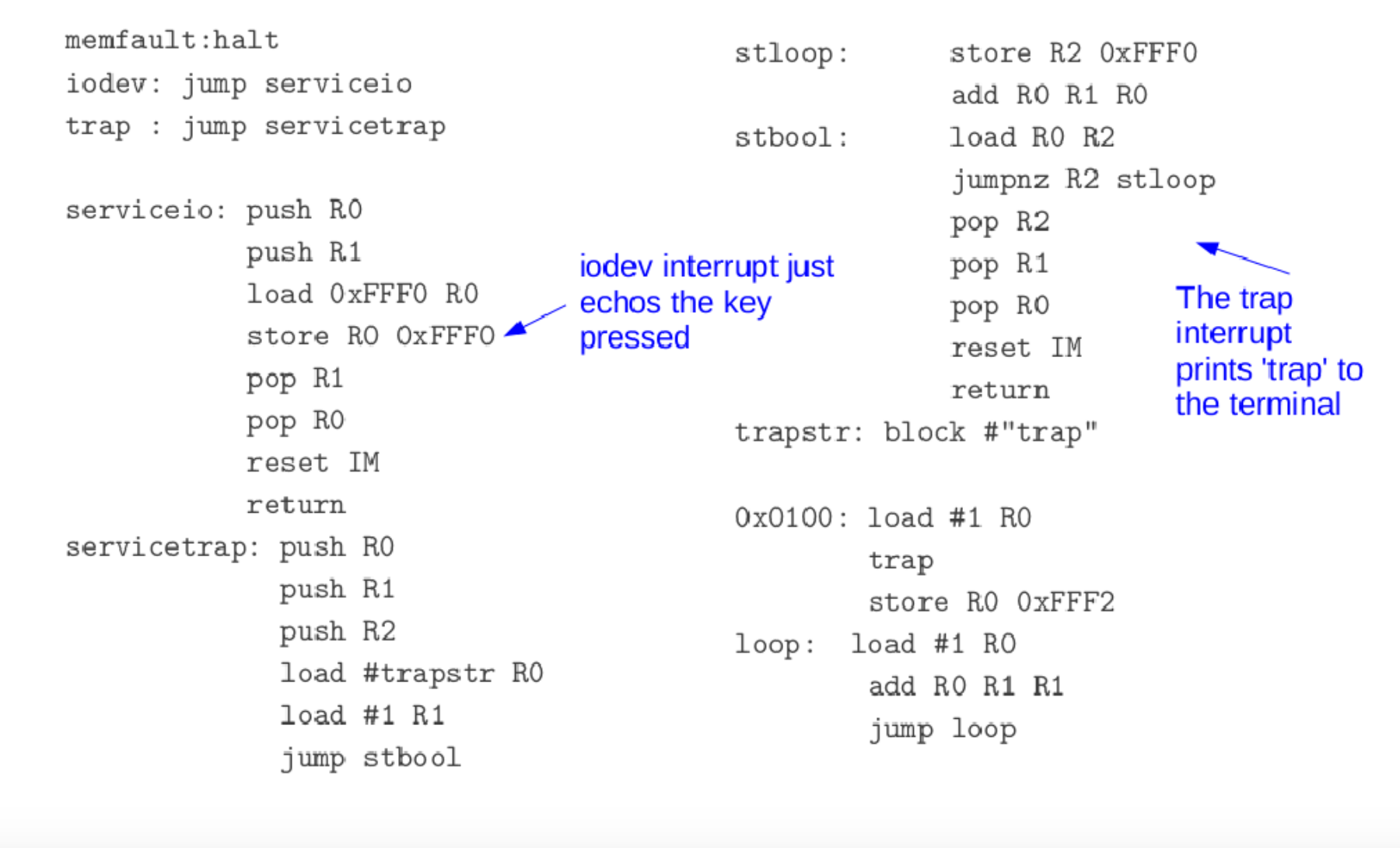
iodevhandler : push R0

load 0xFFF0 R0

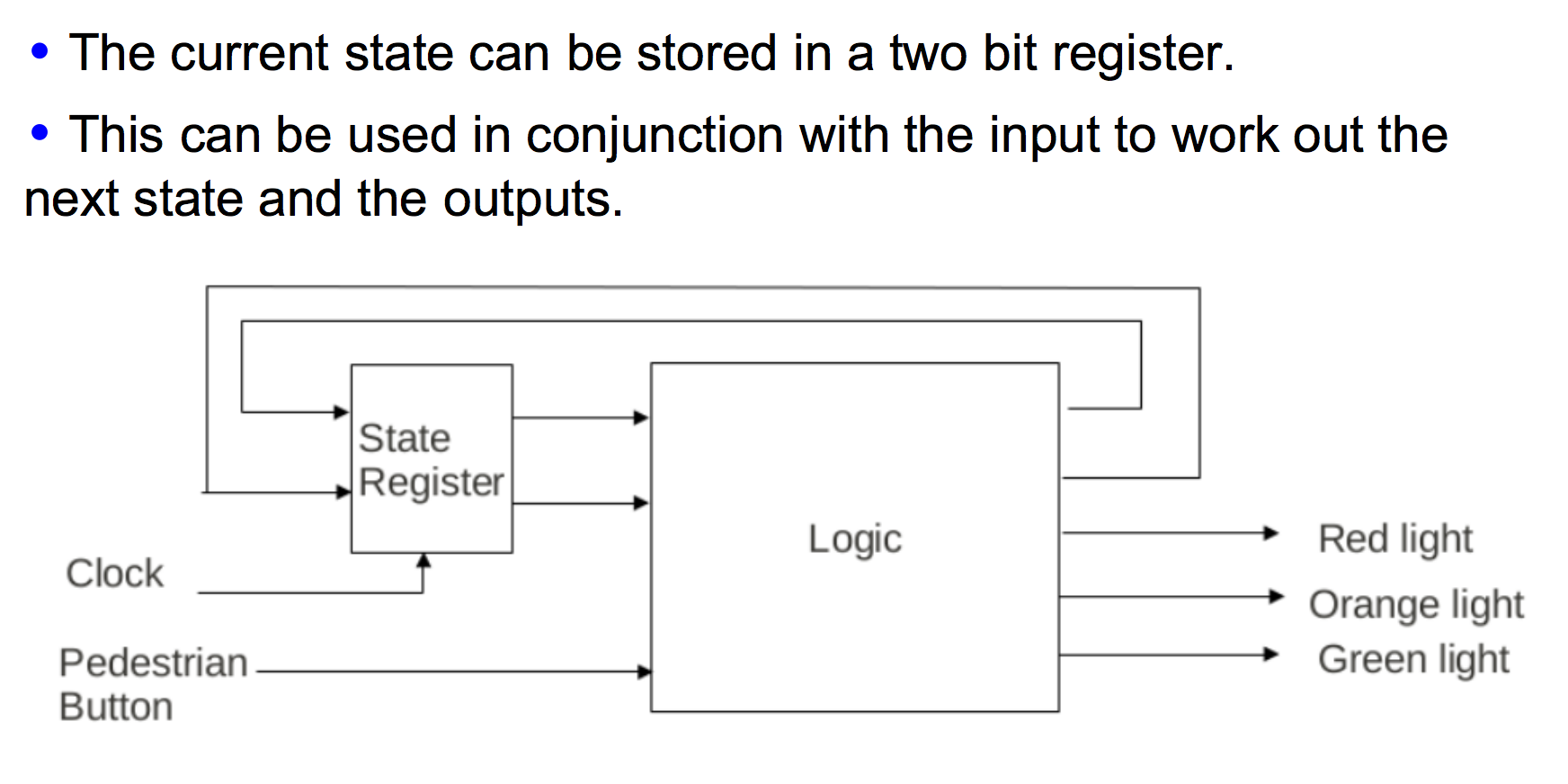
store R0 0xFFF0

pop R0

reset IM return



W10 2:



**W11:**

**Process,thread,program:**

CPUs can maintain the programming context of multiple threads (so duplication of state and register information), without the duplication of processing units, caches, TLBs, etc, this enables multiple threads to be executed within the one core. This can hide latency. So while one thread is waiting on a result another can be executing. Switching between threads is very cheap as it is all done in hardware.

A process is a program in execution. To provide multiprogramming the operating system will maintain multiple independent program contexts. The operating system will manage the simultaneous execution of programs. A process is program in one of these execution contexts. Processes are the basic unit of work in an operating system. A process is an active entity.

**1.**Processes are useful as they effectively allow the computer system to perform many tasks at once. This is useful for a variety of reasons including: Facilitates multi-programming which improves CPU utilisation. Because while programs are waiting on I/O events, other programs may run. Facilitates time-sharing. (Each user gets a slice on the CPU resource.) The user may run different programs at once. (text editor, database, calculator, compiler, etc.) Helps modularity in program design.

A process is a program in execution. It is described by the following information: Program Code (Referred to as the text). Program Counter (or instruction pointer). The next instruction in the text to execute. Scheduling information. Stack. This contains the local variables of the program. Registers. Data Section. The global variables. IO information. (e.g. open files) Process State. Accounting information. This may be used to help record and control the resources used by this process.

**2.**A thread is the basic unit of execution within process. Simple programs normal just have one thread of execution running. However, it is sometimes useful for a program to have multiple units of execution within the program. An OS many provide this resource (kernel threads), or it may be done within user space (user-level threads). A process (or task) consists of resources such as: open files, the text, data section, signals, and a set of threads. Each thread, sometimes called a lightweight process (LWP), consists of a program counter, a register set, and a stack space. A traditional process is called a heavyweight process. heavyweight process = task with one thread.

A thread is the basic unit of CPU utilization. A group of threads combine together to form a task/process. These threads share text, data, open files, capabilities (protection), etc

**Out of order execution:**

Sometimes instructions will require data from memory before they can execute, this will stall the pipeline. This can slow the CPU down greatly.

The "Outer of order execution" approach loads the next few instructions and starts executing the instruction that has the required data, this means instructions may be executed "out of order".

Often there is dependencies between instructions, the CPU must be mindful of these.

**W12:**

**CPU cache:**

A CPU-cache stores copies of main memory data, the aim of this is to reduce the average access time. This works because the cache is small and fast (SRAM) in comparison to main memory which is big and slow (DRAM). Cache's generally store a contiguous 'line' of bytes form memory (typically 64 or 128 bytes). A CPU-cache is critically for the overall performance of a modern computing system. This is because a cache provides a fast way of getting access to the memory it is currently using (~3 clock cycles to access L1 cache, ~15+ L2 cache, ~200+ main memory). The working set is the memory that a program is currently using. Over the execution of a program the working set will change. If the working set fits within the cache then the program will not be slowed by latency to DRAM.

When reading via the cache if the data is in the cache then it can be read immediately. If the data is not in the cache it needs to be loaded from memory into a cache line. This will involve finding and allocating a cache line. This may also involve writing that cache line out. A cache hit happens when accessing data a copy of the memory data is in the cache (so no need to go to main memory). A cache miss happens when on access there is no copy in the cache. This requires access to main memory. A write miss can also happen when a write occurs with data that is not currently cached.

The write policy determines the approach taken when data is written to memory via the cache. There are two basic approaches taken when a write is done: write-through - data is written both to the cache and the memory at the same time. write-back - data is just written to the cache. Another issue is when writing is if the data being written is not in a cache line (write miss) then either: the data could be written directly to memory (no-write allocate) or first loaded into a cache line then written to memory (write allocate).

**Memory management and paging:**

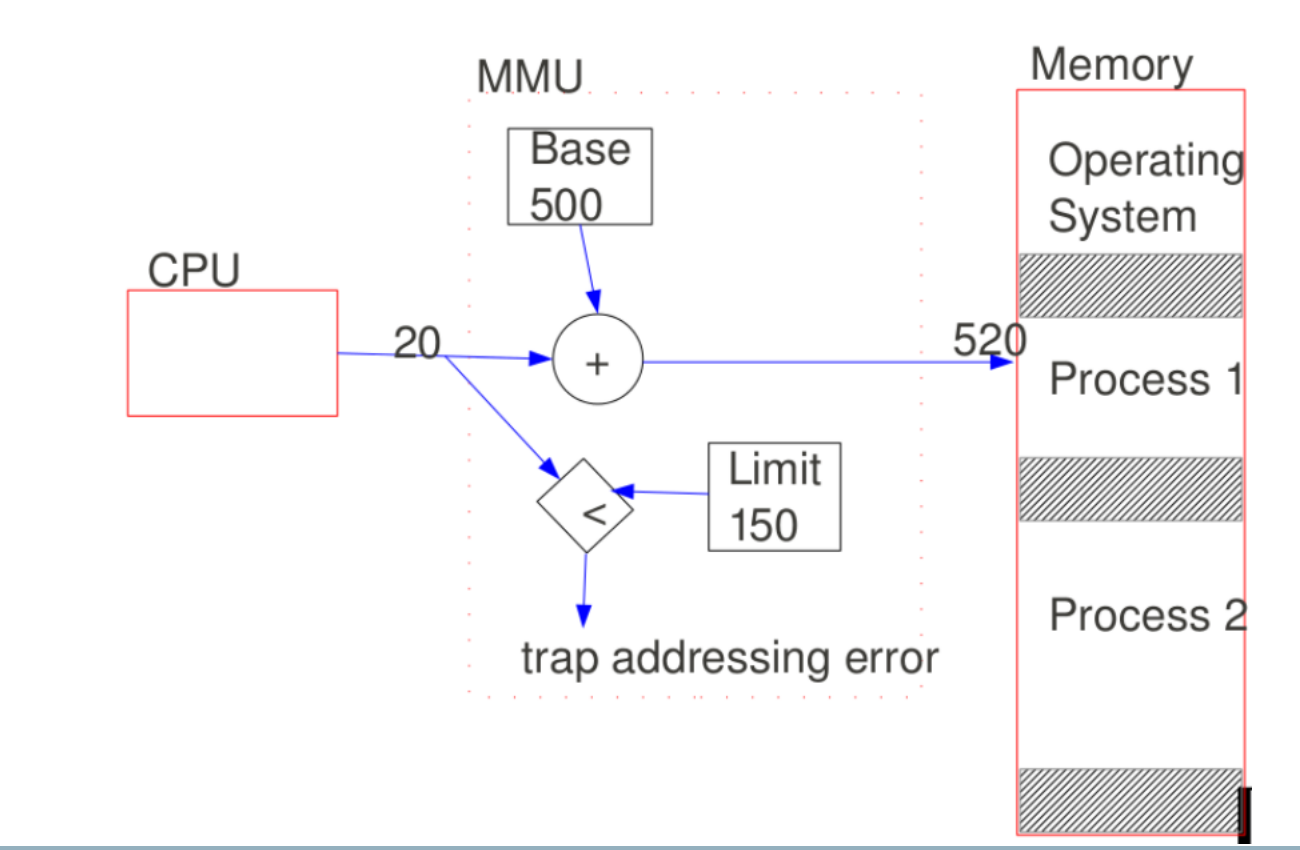
Address Binding

For a program to execute it must be copied into main memory at a particular location. Many instructions use 'fixed' addresses which must be bound to 'fixed' locations in the memory. This binding of instructions and data to memory addresses may occur at : compile time, load time, or execution time.

3 Logical/Physical Address Space Addresses generated by the CPU are referred to as logical addresses. These are the addresses 'seen' by the user's programs. Addresses seen by the main memory are referred to as physical addresses. In some systems logical and physical addresses are identical. In these cases address binding must occur at compile-time or load-time. However, it is useful to separate logical and physical addresses, this permits execution-time address binding schemes. Logical addresses may also be referred to as virtual addresses.

4 Logical/Physical Address Space The set of all logical addresses generated by a program is referred to as the logical address space. These logical addresses map to physical addresses and are referred to as the physical address space. The mapping between the logical and physical addresses is performed by the MMU (Memory-Management Unit). This is done in hardware. Hardware MMU can also provide a range of protections.

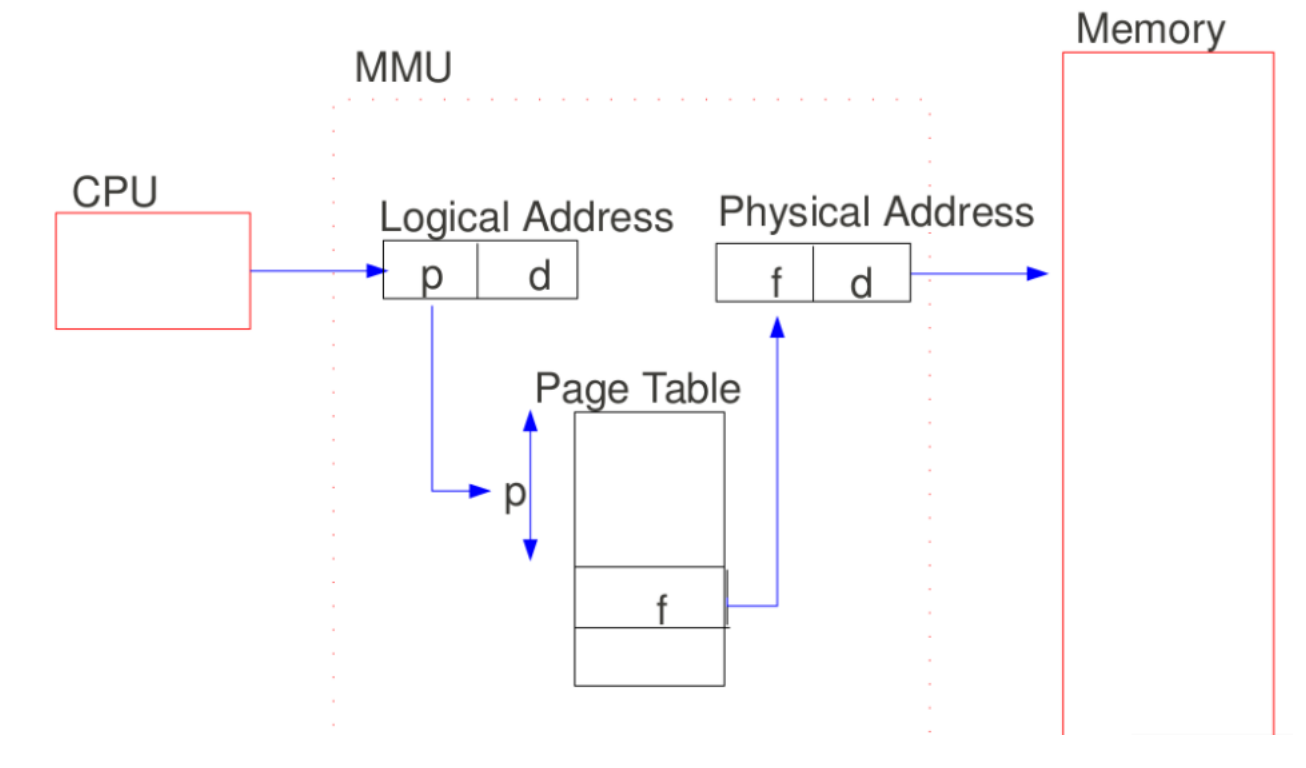
5 Logical/Physical Address Spaces Some advantages of separating logical and physical addresses include: execution-time address-binding, simplifies the swapping of processes in and out of memory, protection/security of data between processes, and simplifies sharing of data.

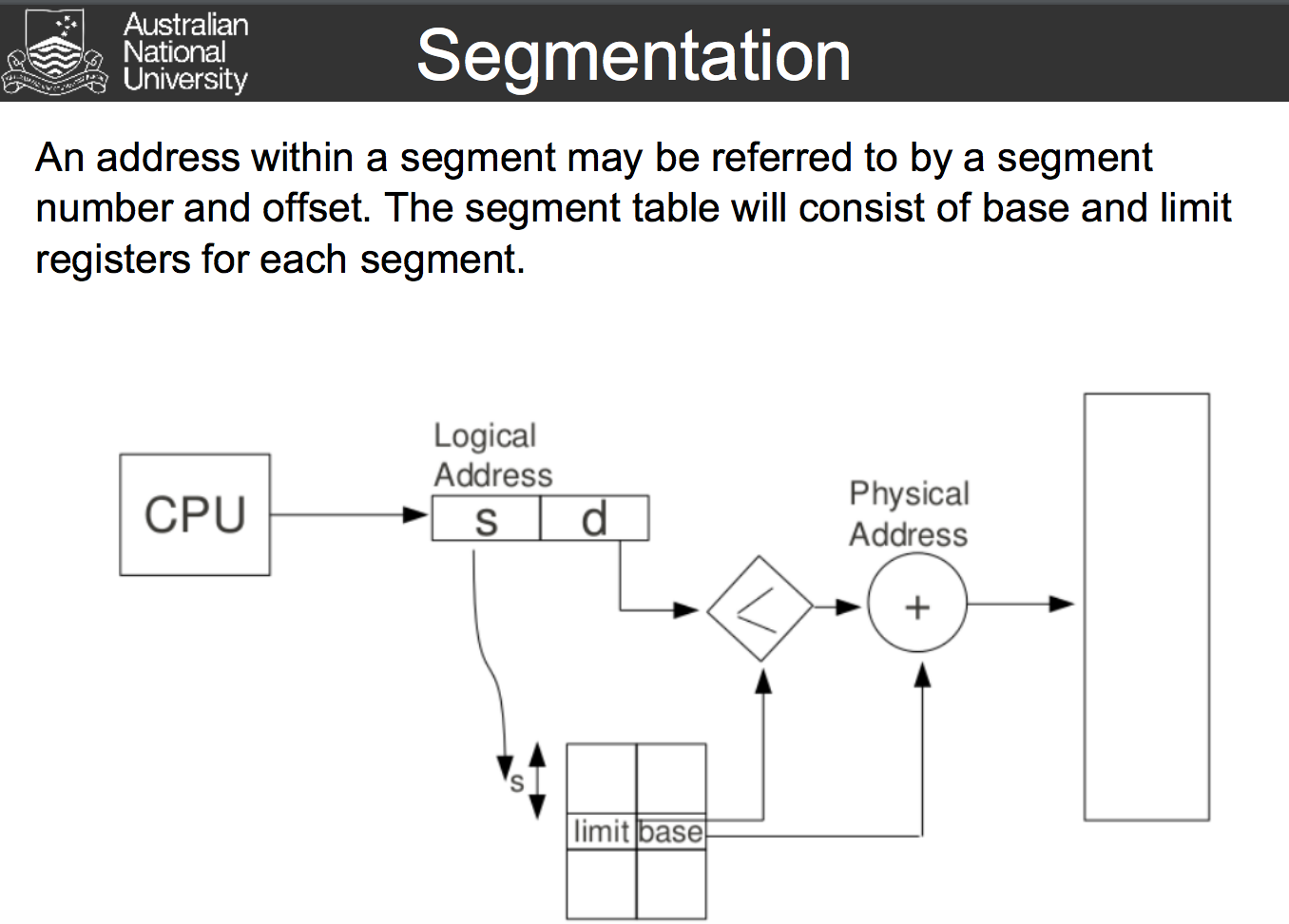


**Paging:**

Paging There is considerable overhead in managing variable sized memory-chunks. Paging overcomes many of these problems and solves the external fragmentation problem (memory gets fragmented up into many small unusable sections). Paging is used in many modern operating systems.

8 Paging Paging involves the following: Physical memory is partitioned into fixed-sized blocks called frames. Logical memory is partitioned into blocks of the same size called pages. Logical addresses (produced by the CPU) are divided into two parts: the page number and the offset. Each process has a page table. The page number indexes the page table for the running process and looks up the frame number for that page. The frame number is combined with the offset to produce the physical address.

****

****

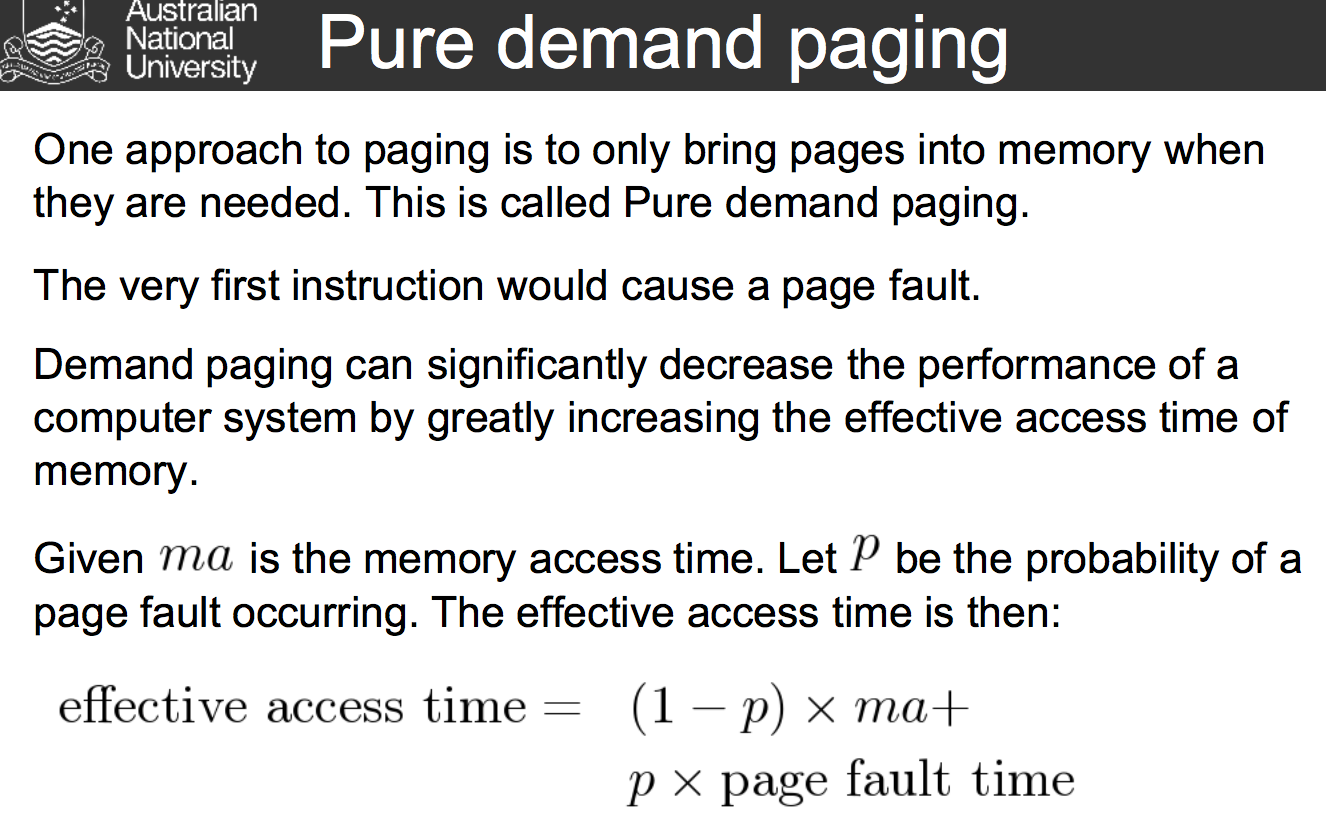
The mapping between logical and physical addresses is done by the MMU(memory management unit) This is done in hardware within the CPU.

**Virtual memory:**

Virtual memory is a technique that permits processes to be executed even when they are not completely in memory. This has many advantages, including: 1.programs can be larger than physical memory,

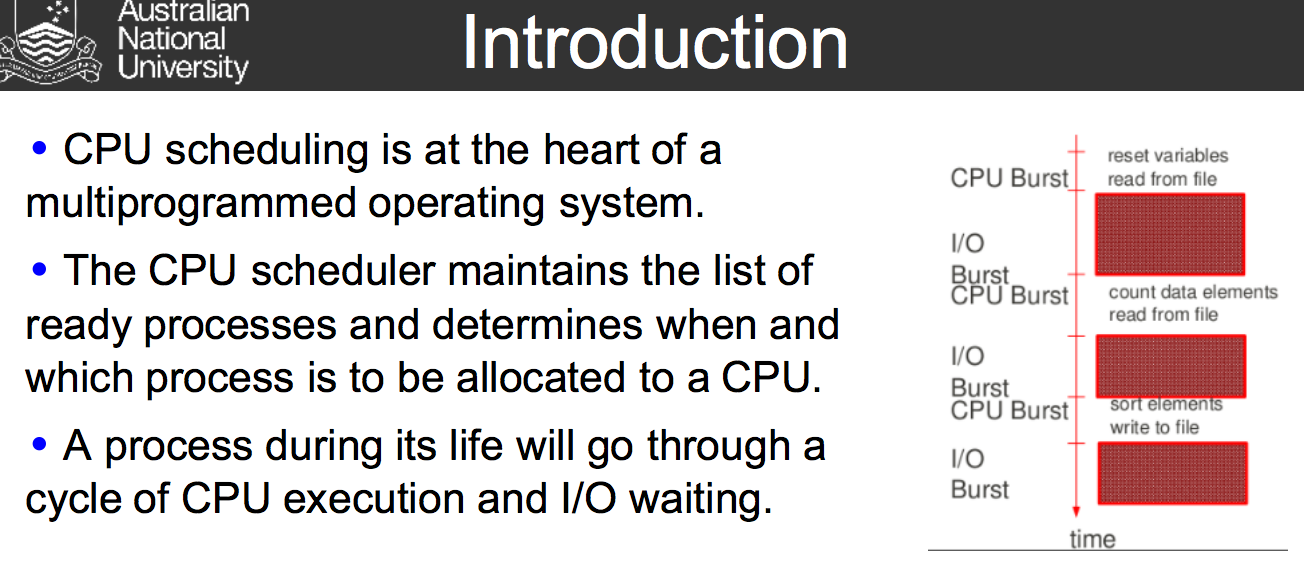
2.virtual memory abstracts main memory into an extremely large logical storage area, and

3.virtual memory increases the degree of multi-programming. However, it is complex to implement and can dramatically decrease performance if it is used carelessly.

****

**Page-Replacement Algorithms** The goal of the page-replacement algorithm is to minimise the page-fault rate. Different algorithms may be compared by computing the number of page faults on a particular reference string. Given the overhead of a page fault, small improvements in the page replacement algorithm will greatly improve the performance of the entire system.

**CPU scheduling:**

****

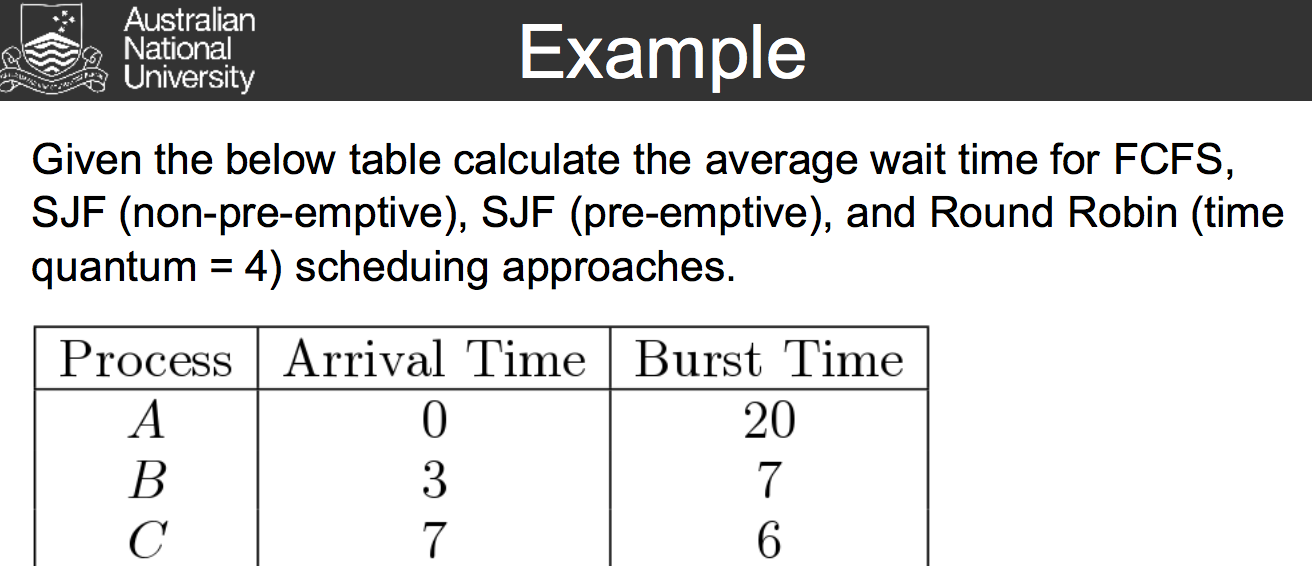
There are two types of schedulers: non-preemptive - The CPU is not 'forcefully' taken from the process, and a preemptive - The CPU may be forcefully taken from the process and switched to another process.

Scheduling Criteria A variety of criteria are used to compare different scheduling algorithms. These include: CPU utilization - The percentage of time the CPU is in use. Throughput - The rate at which processes are being completed. Turnaround time - The interval of time from the starting a process to completing the process. Wait time - The amount of time a process spends in the ready queue. Response time - The amount of time it takes a process to start responding. This does not include the time to output the response.

5 FCFS Scheduling The simplest scheduling algorithm is the First Come First Serve (FCFS) scheduler. It works by maintaining a FIFO queue. When the running process has completed its CPU burst the next process on the ready queue is dispatched. Processes that are either new or have completed their IO burst are placed on the end of the queue. A disadvantage of a FCFS scheduler is IO bound processes will build up behind CPU bound process. This is known as the convoy effect. This will reduce both IO and CPU utilisation.

6 SJF Scheduling The shortest-job-first (SJF) algorithm schedules the process with the shortest next CPU burst time. If the next CPU burst time of two process is identical then a FCFS scheduler will be used to break the tie. The SJF scheduler is provably optimal with respect to minimizing average waiting time for a set of processes. There is generally no way of knowing the length of the next CPU burst. Hence, it is impossible to implement a SJF CPU scheduler.

7 Round Robin Scheduling A round robin scheduler is a FCFS scheduling where process are preempted after a fix time slice (or time quantum) and placed at the end of the FIFO queue. The ready queue may be thought of as a circular queue. The average wait time for a RR scheduler is comparatively longer than SJF scheduling. A round robin scheduler is useful for time-sharing systems. If the time quantum is infinite then RR scheduling is the same as FCFS scheduling. Determining the size of the time quantum is critical. The smaller the time quantum the greater the context switching overhead.

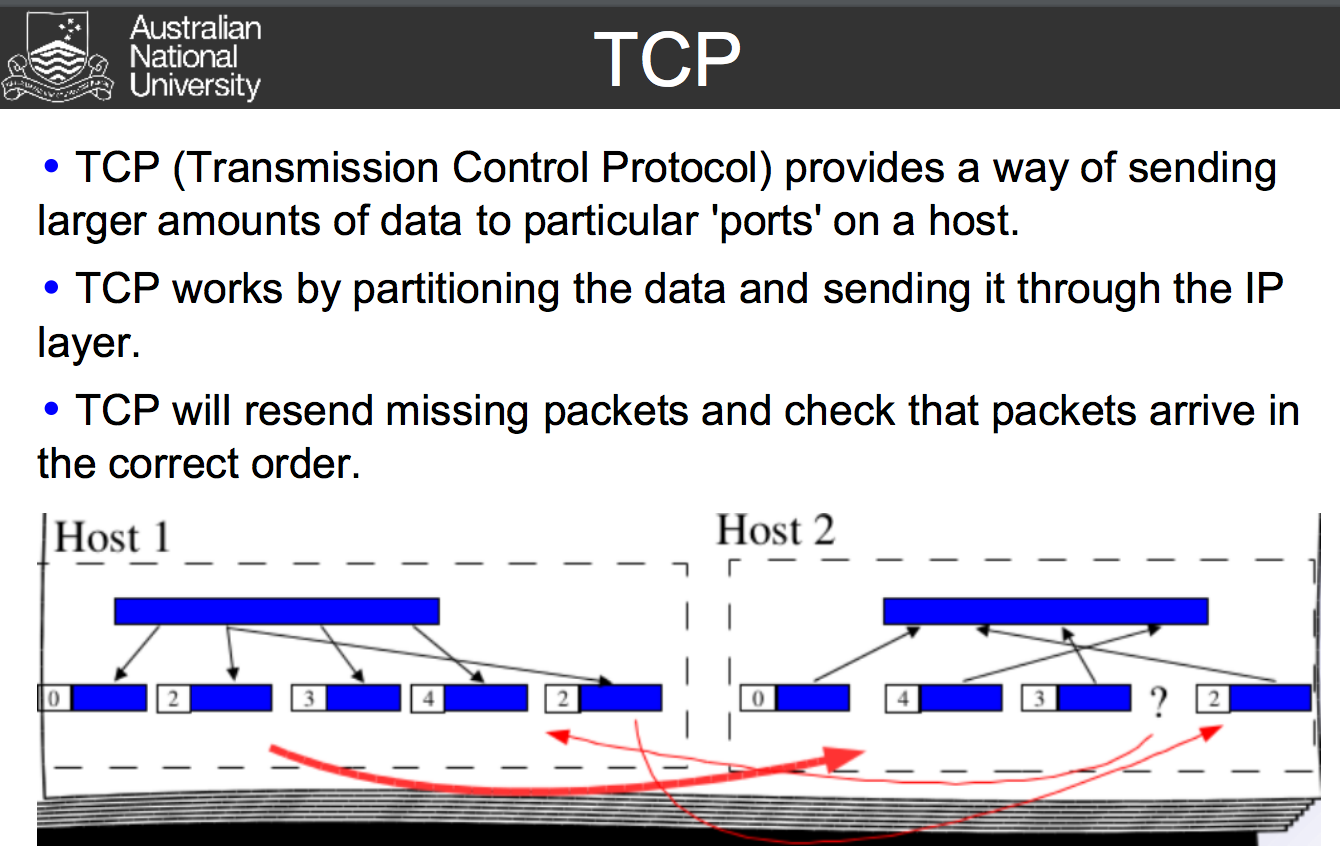
****

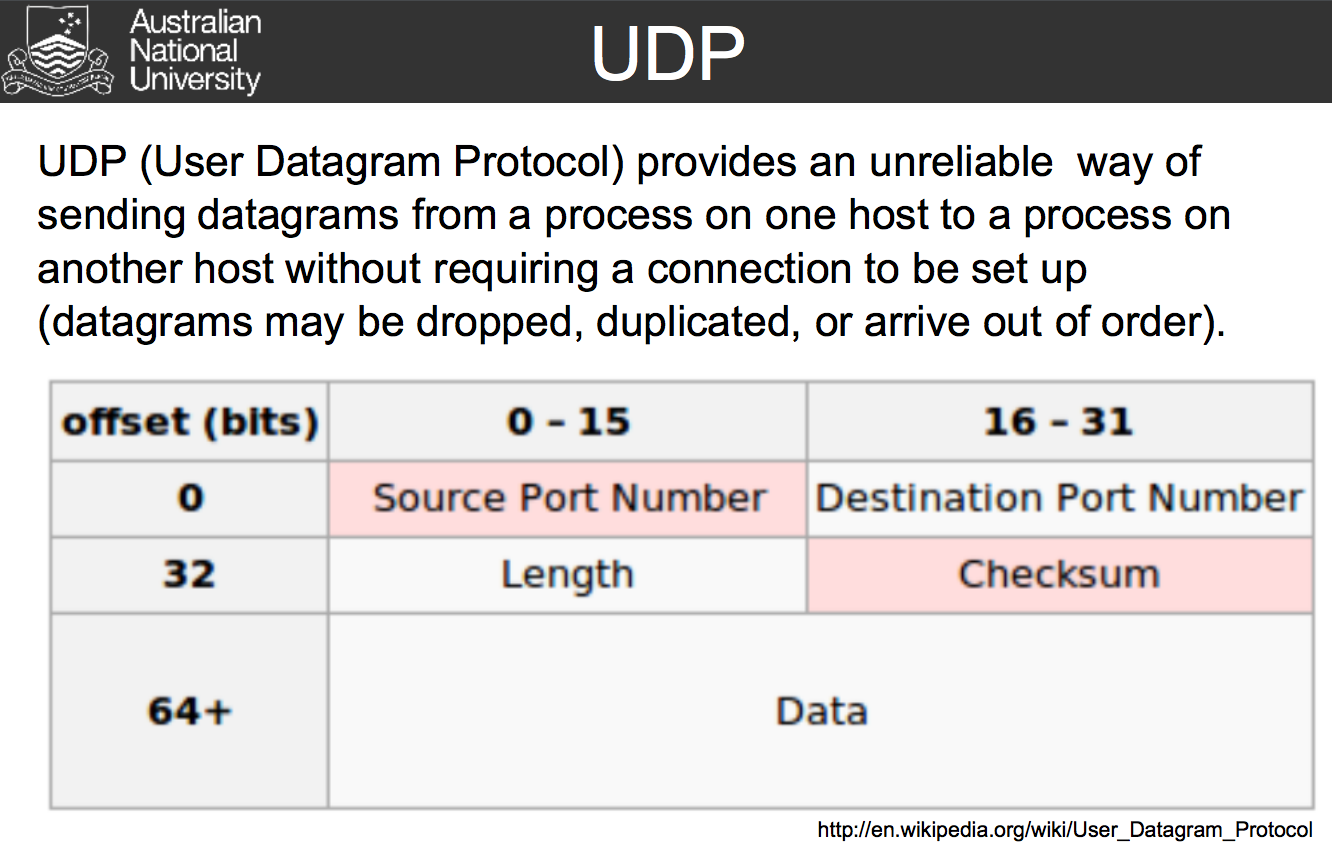
**TLB:**

A translation lookaside buffer (TLB) is a cache that memory management hardware uses to improve virtual address translation speed.[1] The majority of desktop, laptop, and server processors includes one or more TLBs in the memory management hardware, and it is nearly always present in any hardware that utilizes paged or segmented virtual memory.

The TLB is sometimes implemented as content-addressable memory (CAM). The CAM search key is the virtual address and the search result is a physical address. If the requested address is present in the TLB, the CAM search yields a match quickly and the retrieved physical address can be used to access memory. This is called a TLB hit. If the requested address is not in the TLB, it is a miss, and the translation proceeds by looking up the page table in a process called a page walk. The page walk requires a lot of time when compared to the processor speed, as it involves reading the contents of multiple memory locations and using them to compute the physical address. After the physical address is determined by the page walk, the virtual address to physical address mapping is entered into the TLB. The PowerPC 604, for example, has a two-way set-associative TLB for data loads and stores.

**Networks** A communication networks provides the means by which computers can transfer information to and from other computing devices. The Internet has become the most extensive network for computers to communicated across the world. The Internet uses the TCP/IP protocol. This is a packet switching approach. An intranet generally uses the same protocol as the Internet, however, it is focused on communication within an organization. A LAN (Local Area Network) services a limited local area, also generally uses the Internet protocol.

****

****

**CODE:**

**Rpeanut: func(func(m,2\*n),n)**

; stack frame:

; return address #0

; varm #-1

; varn #-2

; result #-3

func:

load SP #-1 R0

load SP #-2 R1

mult R1 #2 R2

sub R0 R2 R3

jumpn R3 funcelseif

push R7

push R2

push R0

func2:

call func

pop R0

pop R2

pop R7

store R7 #-1 SP

jump func

funcelseif:

sub R0 R1 R3

jumpn R3 funcelse

store R3 #-3 SP

return

funcelse:

store R0 #-3 SP

return

0x0100 :

load varm R0

load varn R1

push R2

push R1

push R0

call func

pop R0

pop R1

pop R2

add #48 R2 R2

store R2 0xfff0

halt

varm : block #100

varn : block #5

**gcd(m-n,n)only need the result:**

**; stack frame:**

**; return address #0**

**; varm #-1**

**; varn #-2**

**; result #-3**

**funcelse:**

**load SP #-1 R0**

**load SP #-2 R1**

**sub R1 R0 R3**

**jumpz R3 func**

**jumpn R3 funcelseif**

**push R7**

**push R3**

**push R0**

**call funcelse**

**pop R0**

**pop R3**

**pop R7**

**store R7 #-3 SP**

**return**

**funcelseif:**

**sub #0 R3 R3**

**push R7**

**push R1**

**push R3**

**call funcelse**

**pop R3**

**pop R1**

**pop R7**

**store R7 #-3 SP**

**return**

**func:**

**store R0 #-3 SP**

**return**

**0x0100 :**

**load varm R0**

**load varn R1**

**push R2**

**push R1**

**push R0**

**call funcelse**

**pop R0**

**pop R1**

**pop R2**

**add #48 R2 R2**

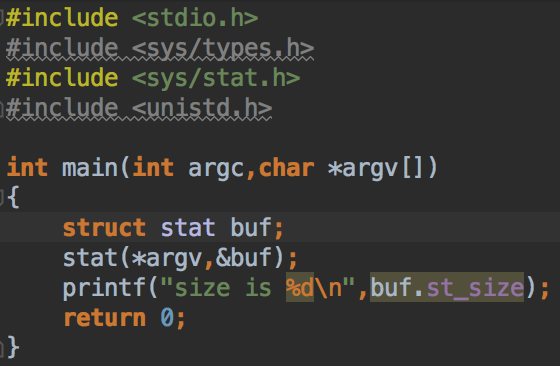
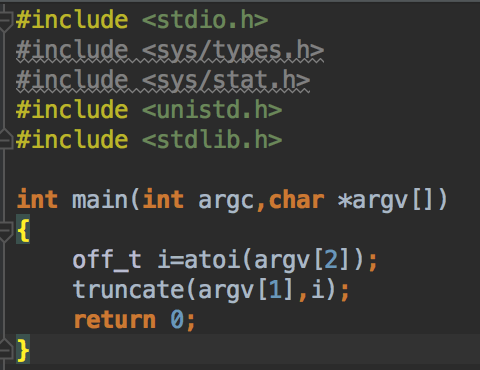
**store R2 0xfff0**

**halt**

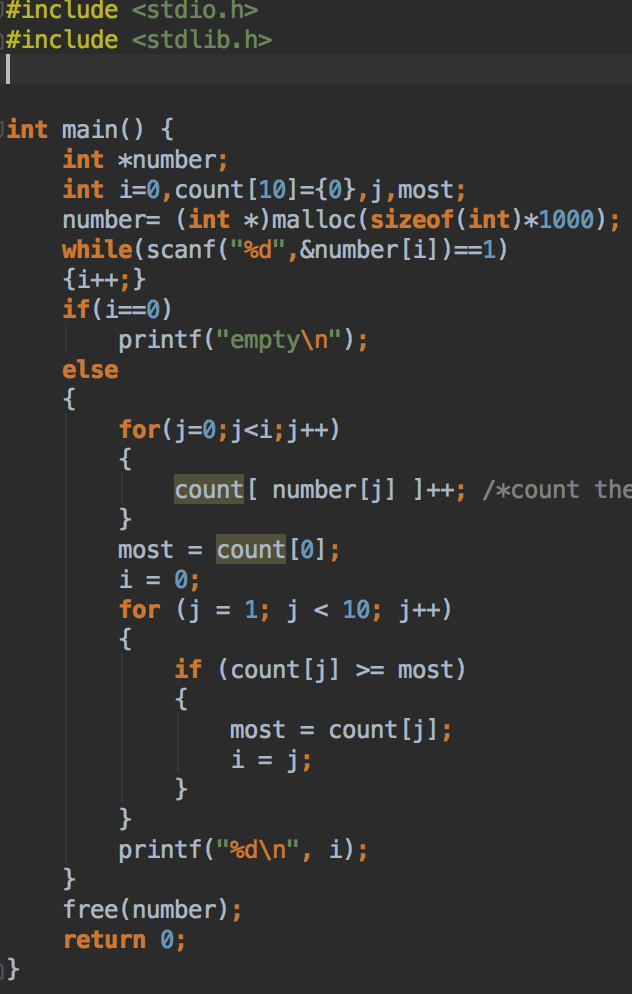
**varm : block #30**

**varn : block #84**

**C: files:**

****

**C numbers:**

****