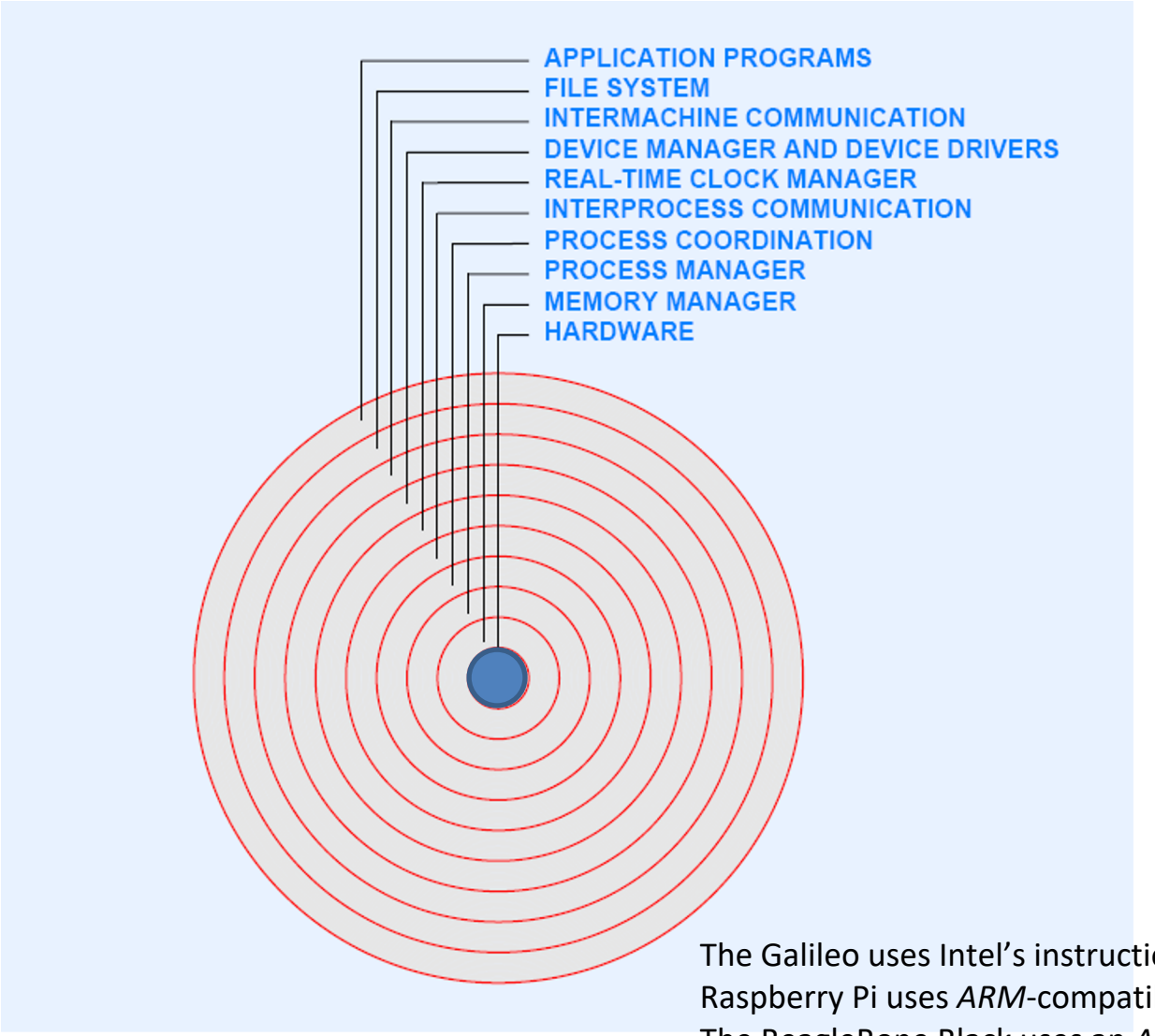
CSCI 8530
Advanced Operating Systems

Part 3

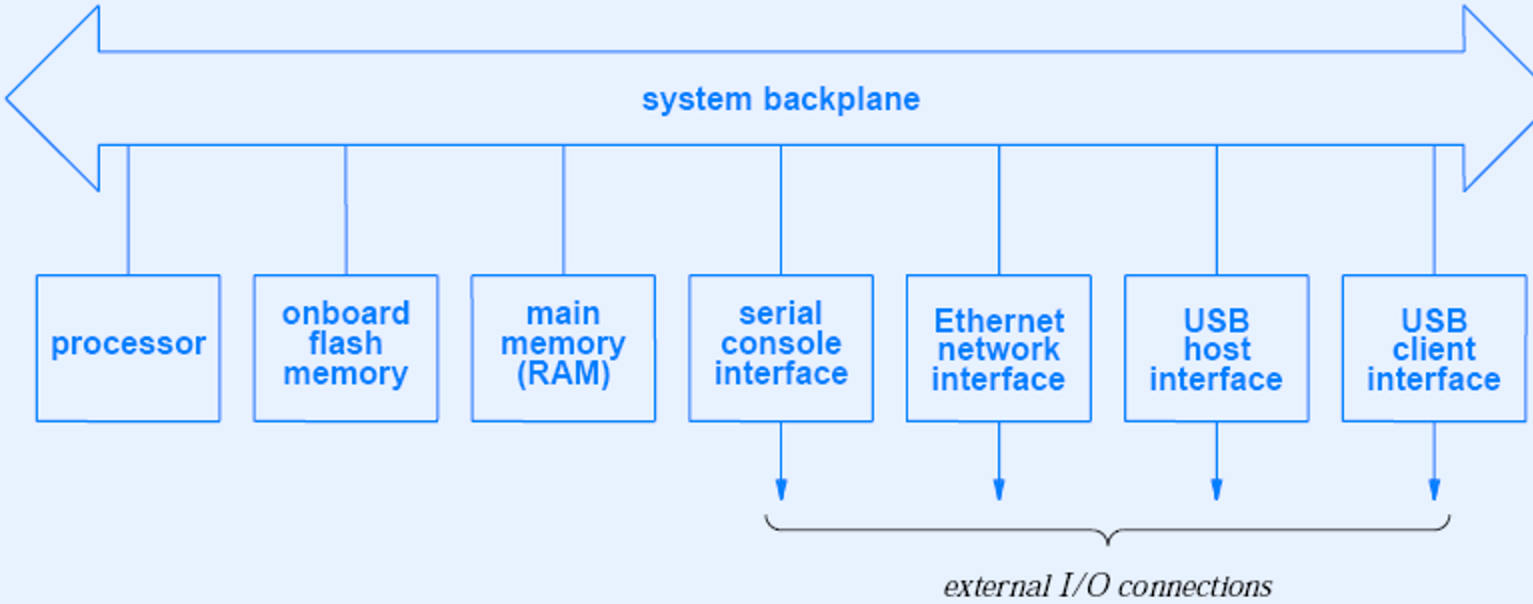
Hardware Architecture and Runtime Systems (A Brief Overview)

Location of Hardware in the Hierarchy



The Galileo uses Intel's instruction set (*x86)*
Raspberry Pi uses *ARM*-compatible CPU and GPU
The BeagleBone Black uses an *ARM* instruction set

Organization of Major Hardware Components

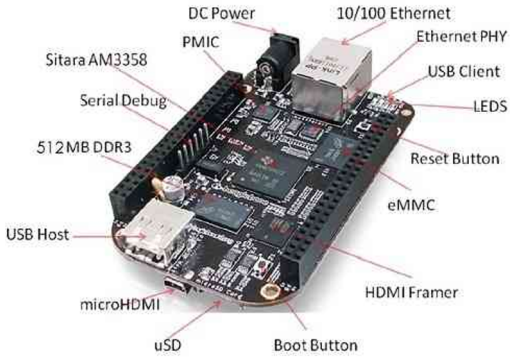


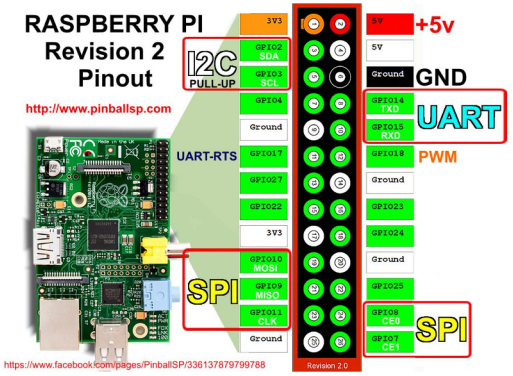
The logical organization of major components in the example platforms

- Xinu has a *serial console* — a character-oriented device uses to issue information messages, report errors, and interact with a user.
- Example platforms use *System on Chip* (*SoC*) approach
- Single VLSI chip contains components and interconnections
- The boards are intended for use in experimental systems, so come completely assembled
- Not relevant to operating system

Key Hardware Features from an Operating System Point of View

- Three example hardware platforms: a Galileo , a BeagleBone Black, a Raspberry Pi
  - Processor
  - Memory system
  - Bus and address space
  - Individual I/O devices
  - Interrupt structure
- The platforms each consist of a small, low-cost circuit board that includes a processor, memory, and a few I/O devices

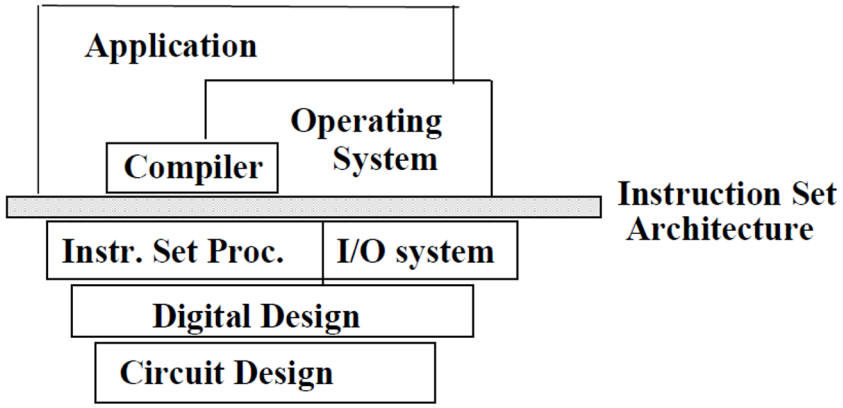




http://kd8bxp.blogspot.com/2015/04/july-26-2014-raspberry-pi.html

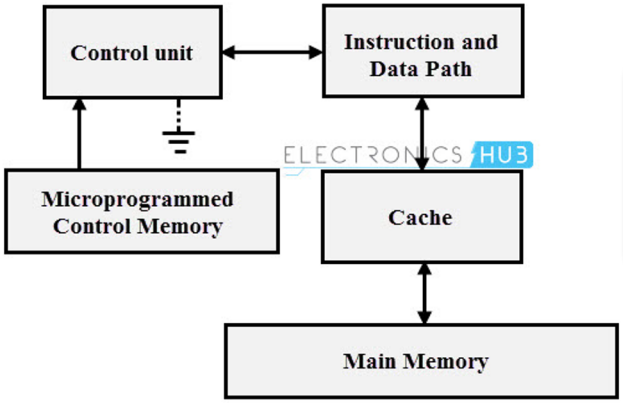
Processor

- Instruction set specifies processor functionality
  - The operations supported by the processor, storage mechanisms of the processor, and the way of compiling the programs to the processor
- Registers: temporary and high-speed storage
- Optional hardware facilities
  - Firmware (e.g., BIOS in ROM)
  - Co-processor
  - Interrupt processor



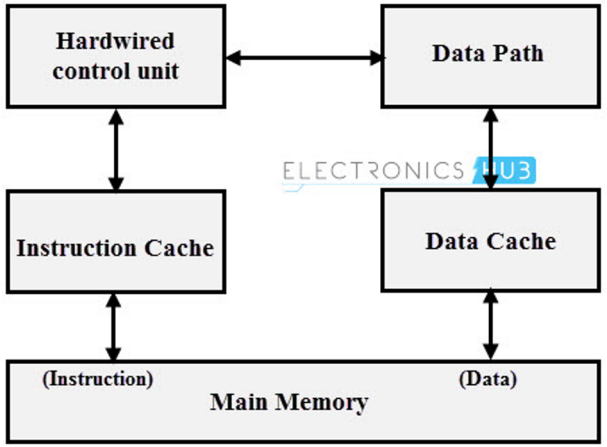
Instruction Sets on the Example Platforms

- Galileo board 2
  - CISC design
  - Well-known Intel (x86) instruction set
- BeagleBone Black
  - RISC design
  - Well-known ARM instruction set
- Raspberry Pi
  - RISC design
  - Well-known ARM instruction set



Access memory less frequently and reduce burden to compiler

CISC: directly use the memory



Use small/highly optimized set of instructions to increase execution speed

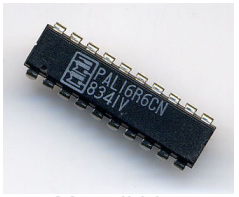
RISC: register to register operations by using small and highly optimized set of instructions

General-purpose and Special-purpose Registers

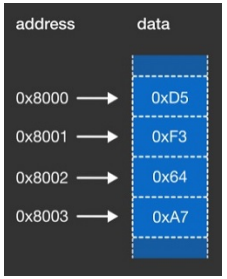
- General-purpose registers
  - Local storage inside processor
  - Hold active values during computation (e.g., used to compute an expression)
  - Saved and restored during subprogram invocation
- Special-purpose registers
  - Located inside the processor
  - Values control processor actions (e.g., mode and address space visibility)

Memory System

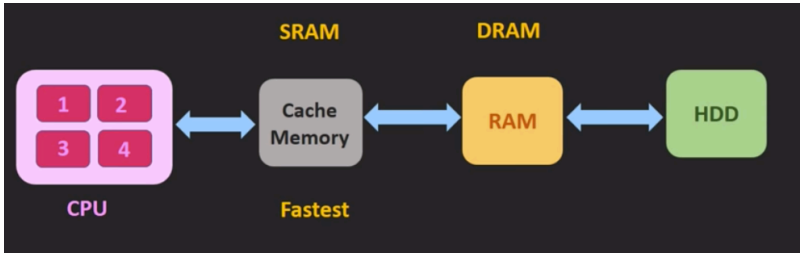
- Defines size of a *byte*, the smallest addressable unit
- Provides address space
- Typical physical address space is
  - *Monolithic*
  - *Linear* (but may not be contiguous)
  - Includes caching
- Defines important property for programmers: endianness



Monolithic Memories, Inc. (MMI)



Structure of a Linear Memory Model



General-purpose Registers

Name	Use
EAX	Accumulator
EBX	Base
ECX	Count
EDX	Data
ESI	Source Index
EDI	Destination Index
EBP	Base Pointer
ESP	Stack Pointer

The general-purpose registers in the Galileo board (Intel) and the meaning of each.

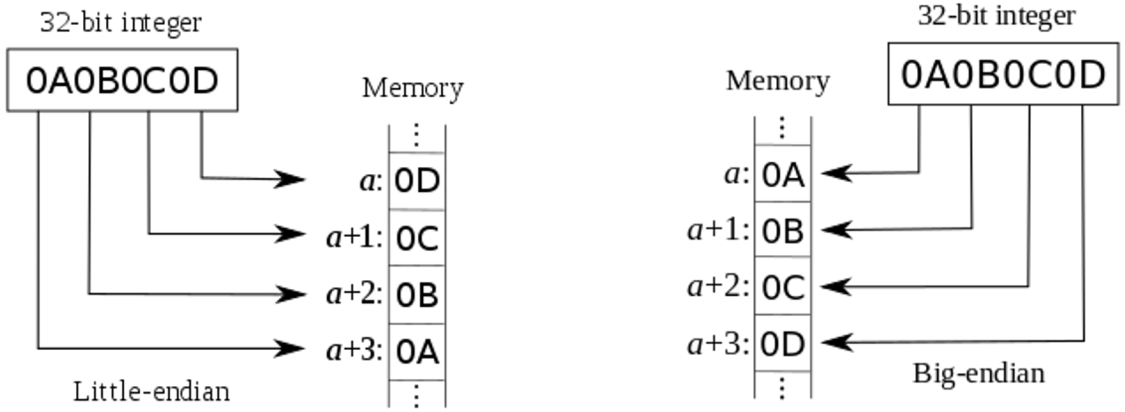
Name	Alias	Use
R0 – R3	a1 – a4	Argument registers
R4 – R11	v1 – v8	Variables and temporaries
R9	sb	Static base register
R12	ip	Intra procedure call scratch register
R13	sp	Stack pointer
R14	lr	Link register used for return address
R15	pc	Program counter

The general-purpose registers and the program counter in the BeagleBone Black (ARM) and the meaning of each.

GPIO in the Raspberry Pi

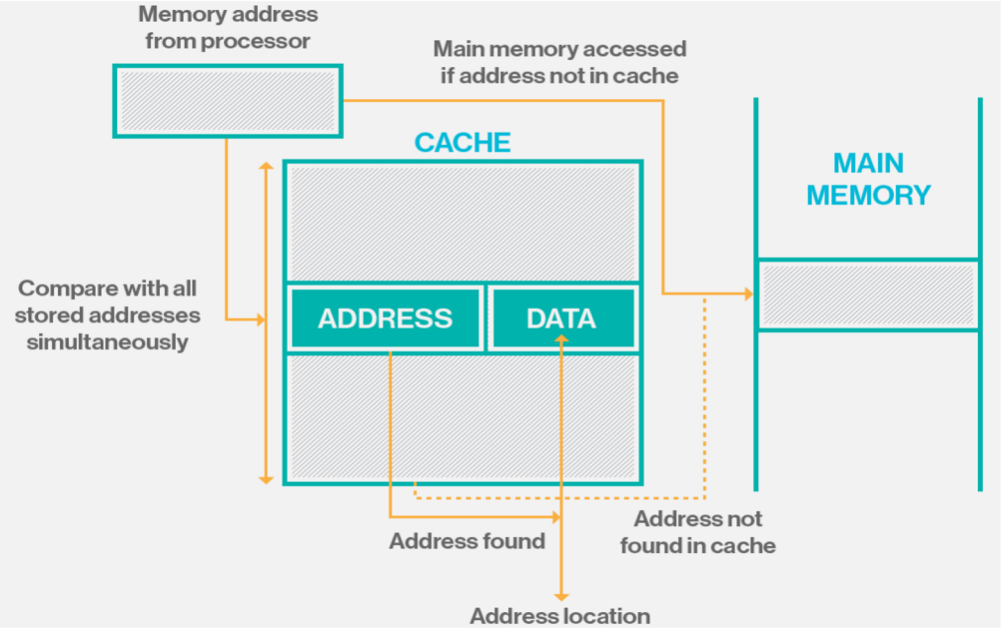
Byte Order Terminology

- Order of bytes within an integer in memory
- Irrelevant to data in registers
- *Little Endian* stores least-significant byte at lowest address
- *Big Endian* stores most-significant byte at lowest address



Memory Caches

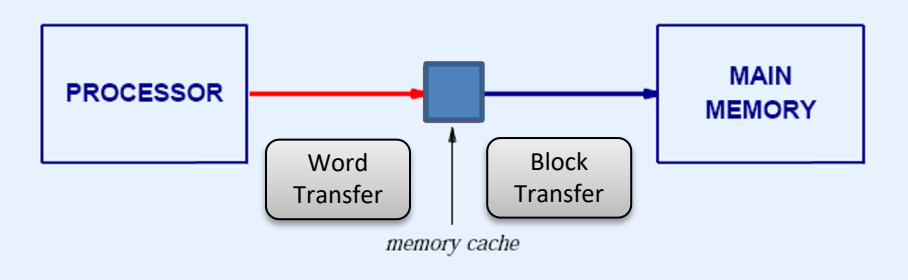
- Special-purpose hardware units
- Speed memory access
- Less expensive than high-speed memory
- Physically associated with memory
- Conceptually placed “between” processor and memory



https://searchstorage.techtarget.com/definition/cache

Conceptual Placement of Memory Cache

- All references (including instruction fetch) go through cache
- Multi-level (L1-L4) cache possible
  - L1 built on the microprocessor chip; fastest but need large spaces (expensive)
  - L2 embedded on the CPU or a separate chip or coprocessor
  - L3 specialized memory developed to improve L1 and L2 performance
  - L4 accessed and shared by the CPU and the GPU.
- In practice, placing the cache on the processor chip improves performance
- Key question: are virtual or physical addresses cached?
  - Translation lookaside buffer (TLB) is memory cache that stores recent translations of virtual memory to physical addresses and speeds up virtual memory operations.
- Consequence: operating system may need to
  - Avoid the cache when accessing a device
  - Flush the cache



I/O Devices

- Wide variety of peripheral devices available
  - Displays
  - Keyboards / mice
  - Disks
  - Wired and wireless network interfaces
  - Printers and scanners
  - Cameras
  - Sensors
- Multiple transfer paradigms (character, block, packet, stream)

Communication Between Device and Processor

- Communication with I/O devices is *memory-mapped*
  - Each I/O device is assigned a set of addresses in the bus address space
- Processor can
  - Interrogate device status
  - Control (e.g., shutdown or awaken) a device
  - Start a data transfer
- Device uses bus to
  - Reply to commands from the processor
  - Transfer data to/ from memory
  - Interrupt when operation completes

A sample system memory map

Address range (hexadecimal)	Size	Device
0000–7FFF	32 KiB	RAM
8000–80FF	256 bytes	General-purpose I/O
9000–90FF	256 bytes	Sound controller
A000–A7FF	2 KiB	Video controller/text-mapped display RAM
C000–FFFF	16 KiB	ROM

https://en.wikipedia.org/wiki/Memory-mapped_I/O

Bus Operations (1/2)

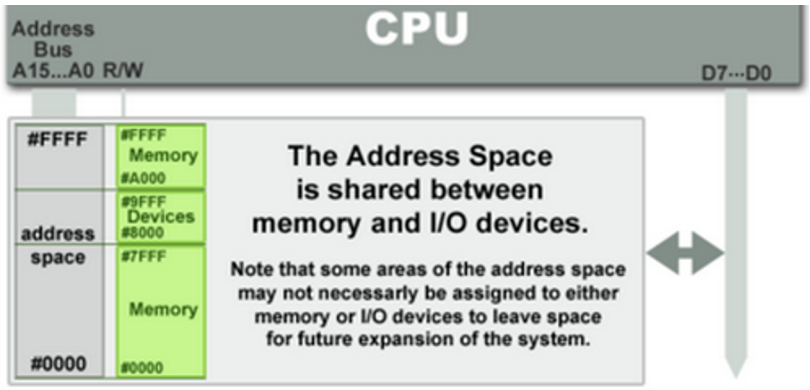
- Only two basic operations supported
  - Fetch: to move data from a component to the processor
  - Store: to move data from the process to a component
- All I/O uses fetch-store paradigm
- Fetch
  - Processor places address on bus
  - Processor uses control line to signal *fetch request*
  - Device senses its address
  - Device puts specified data on bus
  - Device uses control line to signal *response*

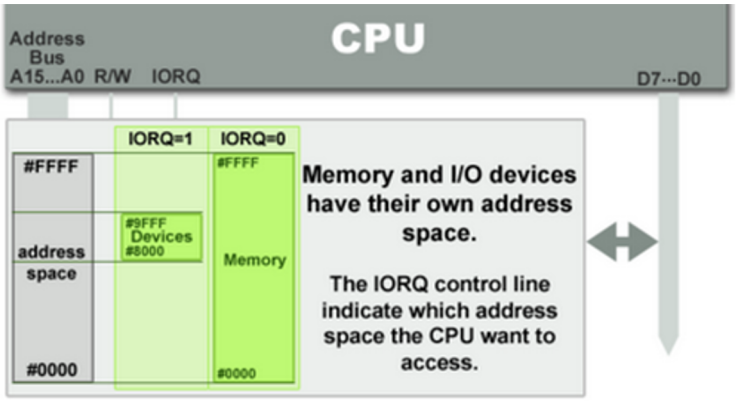
Bus Operations (2/2)

- Store
  - Processor places data and address on bus
  - Processor uses control line to signal *store request*
  - Device senses its address
  - Device extracts data from the bus
  - Device uses control line to signal *data extracted*

Two Basic Approaches for I/O

- *Memory-mapped I/O*
  - Devices placed beyond physical memory
  - Processor uses normal fetch / store memory instructions
  - On later Intel machines as well as most others
  - Mapped into the same address space
- *Port-mapped I/O*
  - Special instructions used to access devices
  - Used on earlier Intel machines
  - Use a separate, dedicated address space

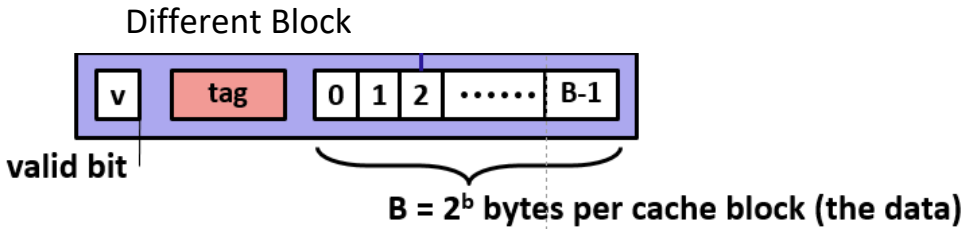




https://www.bogotobogo.com/Embedded/memory_mapped_io_vs_port_mapped_isolated_io.php

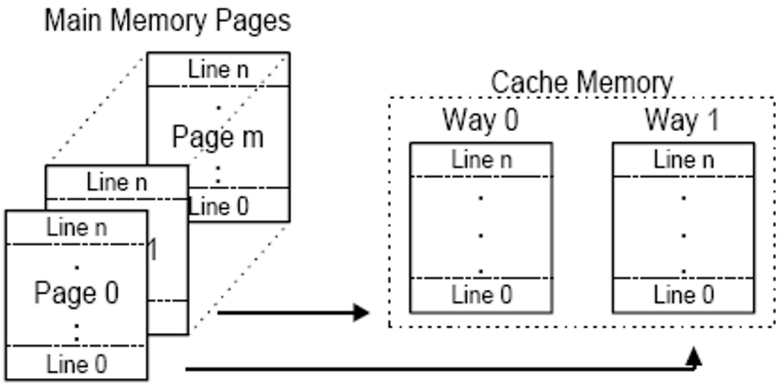
Hardware Cache Types

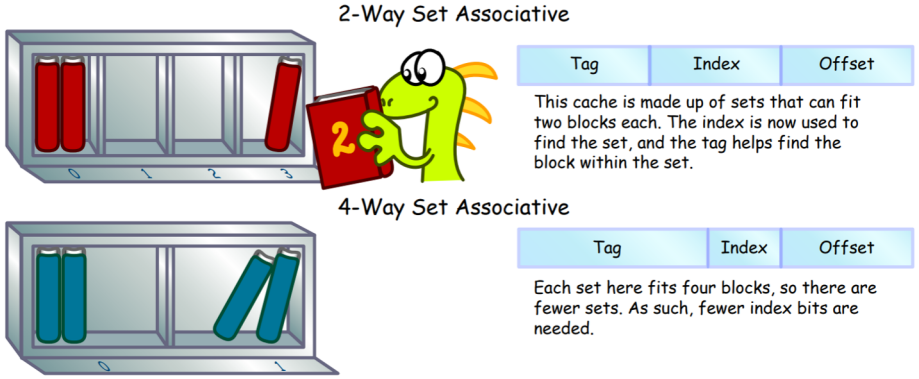
- According to where a memory line (valid bit, tag, and data) is stored in the cache, there are three different types of caches:
  - Degree N-way set associative.
  - Fully associative.
  - Direct mapped.



Degree N-way Set Associative

- A set-associate scheme works by dividing the cache **SRAM** into equal sections (2 or 4 sections typically) called ***cache ways***. The cache page size is equal to the size of the cache way.
- A memory line *w* of a cache page can only be stored in the cache line *w* of one of the ***cache ways***.

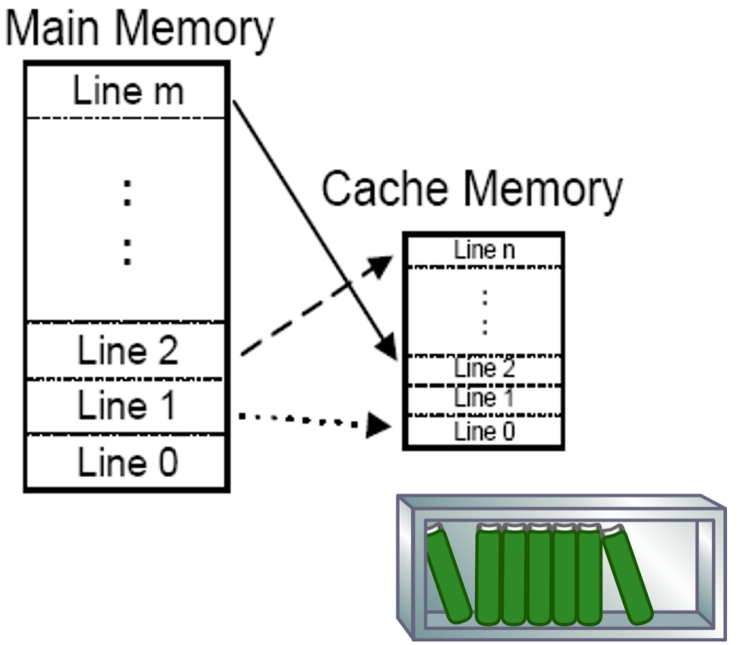




http://csillustrated.berkeley.edu/PDFs/handouts/cache-3-associativity-handout.pdf

Fully Associative

- Main memory and cache memory are both divided into lines of equal size.
- This organizational scheme allows any line in main memory to be stored at any location in the cache.



Fully Associative

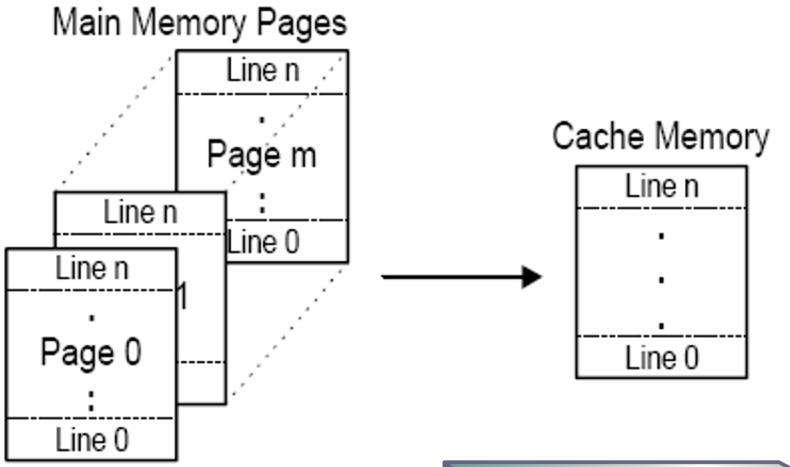
Tag	Offset

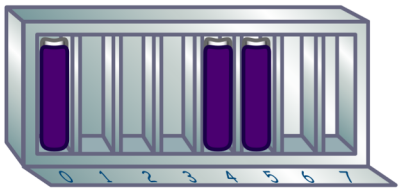
No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!

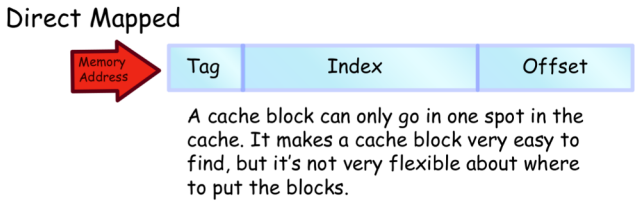
http://csillustrated.berkeley.edu/PDFs/handouts/cache-3-associativity-handout.pdf

Direct Mapped

- Direct Mapped cache is also referred to as ***1-way set associative cache***.
- In this scheme, *main memory* is divided into ***cache pages***. The size of each page is equal to the size of a ***cache***. Unlike the fully associative cache, the direct map cache may only store a specific memory line of a cache page within the same cache line of the cache.





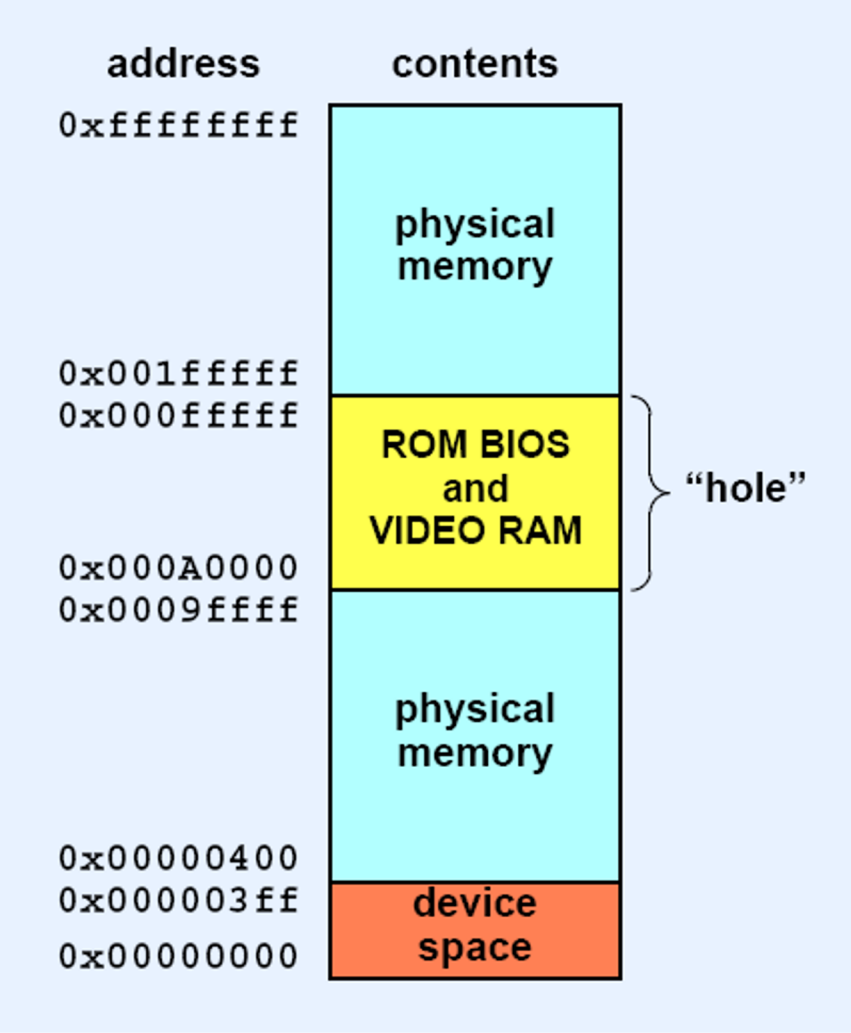


http://csillustrated.berkeley.edu/PDFs/handouts/cache-3-associativity-handout.pdf

Cache Type Summary

- Direct mapped: a memory line is always stored at the same cache line.
- Degree N-way set associative: Cache lines are divided into N sets, a memory line can be in any set of the N sets. But inside a set, the memory line is stored into the same cache line. This is the most popular cache type.
- Fully associative: a memory line can be stored at any cache line.

Address Space on an Intel System



- Traditional PCs have a "hole" from 640KB to 1MB

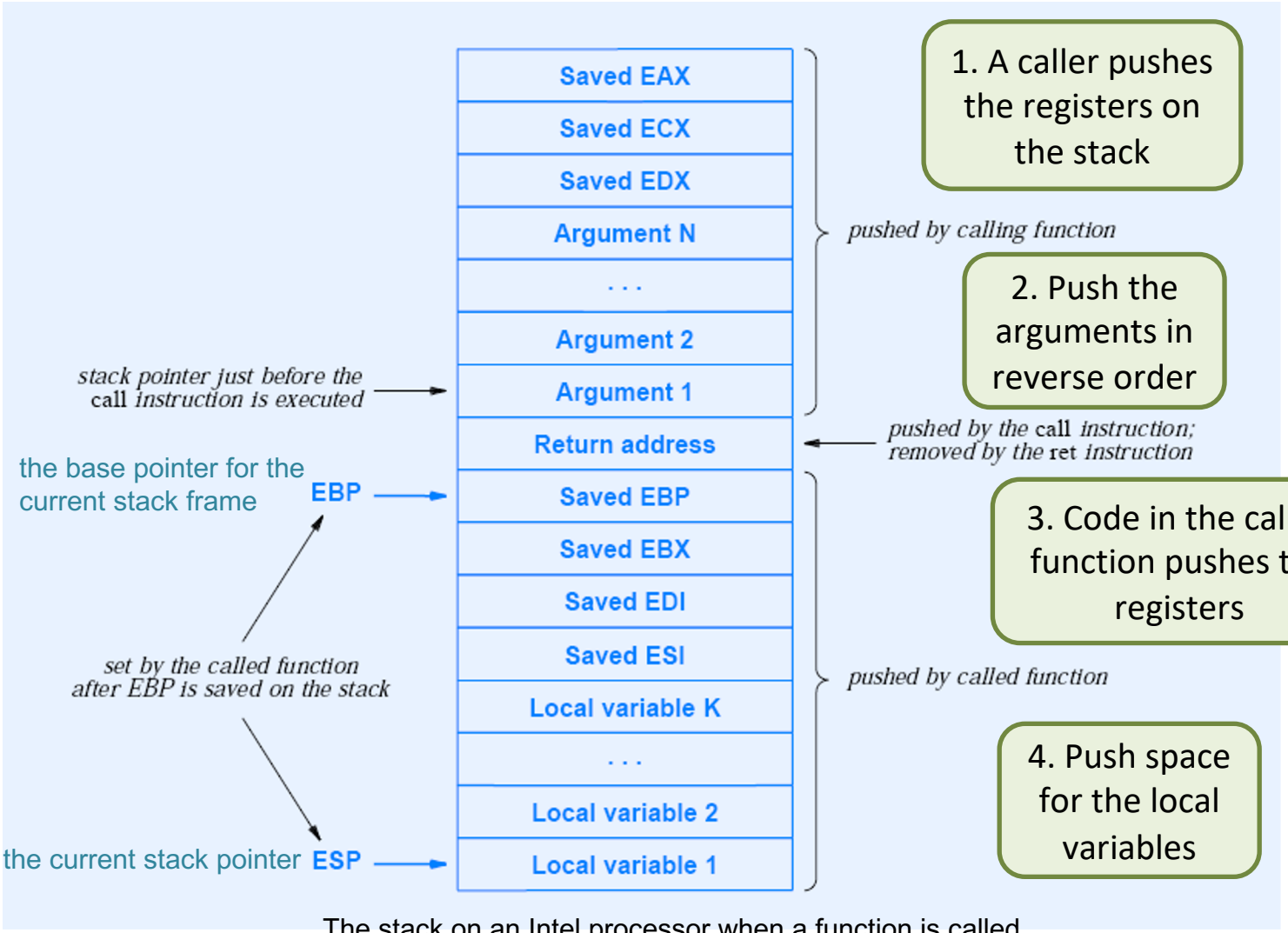
Descriptor Cache Registers *[Robert Collins]*

- Whether in real or protected mode, the **CPU** stores the base address of each segment in hidden registers called ***descriptor cache registers***.
- Each time the **CPU** loads a segment register, the segment base address, segment size limit, and access attributes (access rights) are loaded, or "cached,") into these hidden registers.

Calling Conventions

- Refer to the set of steps taken during a function call
- The conventions specify
  - Which registers must be saved by caller
  - Which registers must be saved by called function
  - Where registers are saved (e.g., on the stack)
- Specific details may depend on
  - The hardware
  - The compiler being used
- The key operating system's functions cannot be implemented unless the calling conventions are known
- When a function is called, the caller supplies a set of actual *arguments* that correspond to formal *parameters*

Calling Conventions for Intel

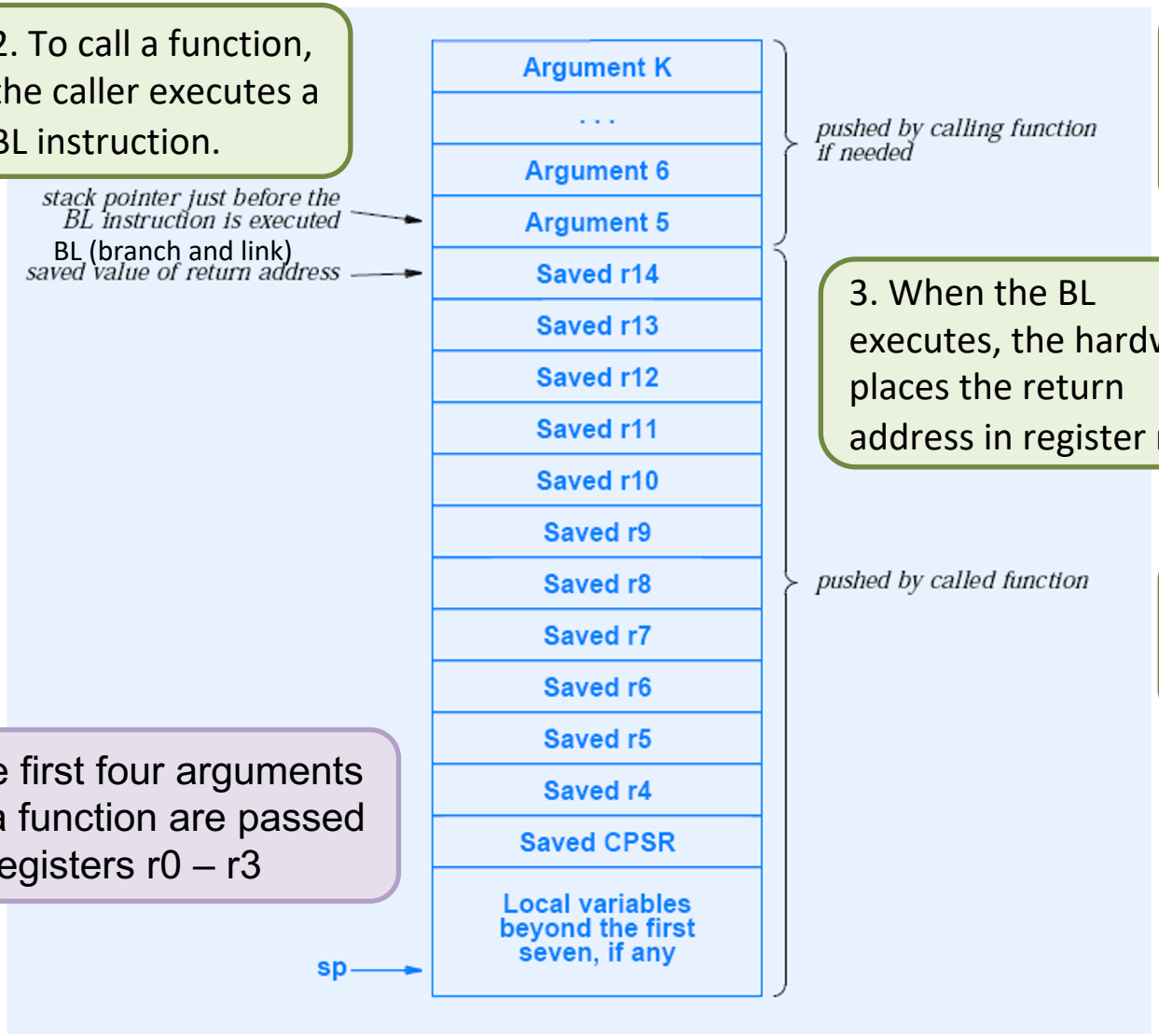


The stack on an Intel processor when a function is called (remember that a stack grows downward).

Calling Conventions for ARM

2. To call a function, the caller executes a BL instruction.

1. Arguments beyond the first four are passed on the stack



3. When the BL executes, the hardware places the return address in register r14.

4. The called function must save registers it will use

The first four arguments to a function are passed in registers r0 – r3

The stack on an ARM processor when a function is called.

ARM Particulars

- Arguments 1-4 placed in registers a0-a3 (r0-r3)
- Other arguments passed on stack (like Intel)
- BL (branch and link) instruction saves return address in r14.
- Called function saves r14-r4 and CPSR (status register) on the stack.
- First 7 local variables in r4-r8, r10, and r11. Others placed on top of stack.
- Different ARM processors may have other registers (e.g. floating point) which must be saved.

Interrupt Mechanism

- Fundamental role in modern system
- Permits I /O concurrent with execution
- Allows device priority
- Processor hardware has a exception mechanism: inform the software when an error or fault occurs
- Informs processor when I /O finished
- *Software interrupt* also possible

Hardware or OS Guarantee …

- When an interrupt occurs, the entire state of the processor, including the program counter and status registers, is saved
- The processor runs the appropriate interrupt handler processor, which must have been placed in memory before the interrupt occurs
- When an interrupt finishes, the operating system and hardware provide mechanisms that restore the entire state of the processor and continue processing at the point of interruption

Interrupt-Driven I/O

- Processor starts device
- Device operates concurrently
- Device interrupts the processor when finished with the assigned task
- Interrupt timing
  - Asynchronous with respect to computation
  - Synchronous with respect to an individual instruction (occurs between instructions)

Interrupt Mask

- The simplest way to prevent other processes trying to manipulate share data: Disabling interrupts.
  - An interrupt mask mechanism: an OS can use to control interrupts
- Determines whether interrupts are enabled
  - The interrupt mask is assigned a value of Zero → the processor ignores all interrupts
- Bit mask kept in a processor status register
- Hardware sets mask during interrupt to prevent further interrupts
- Interrupt priority scheme
  - Offered by some hardware
  - Each device assigned priority level (binary number)
  - When servicing level *K* interrupt, hardware sets mask to disable interrupts at level *K* and lower

Interrupt Processing

- Operating system must
  - Store address of interrupt code in *vector* for each device
  - Creates an array of pointers in memory known as *vectored interrupts*
  - Arrange for interrupt code to save registers used during the interrupt and restore registers before returning
- When interrupts, a device sends its vector number over the bus to the processor
  - Depend on the processor details, either the hardware or the operating system uses the vector number as an index into the interrupt vector, obtains a pointer, and uses the pointer as the address of the code to run

Returning from an Interrupt

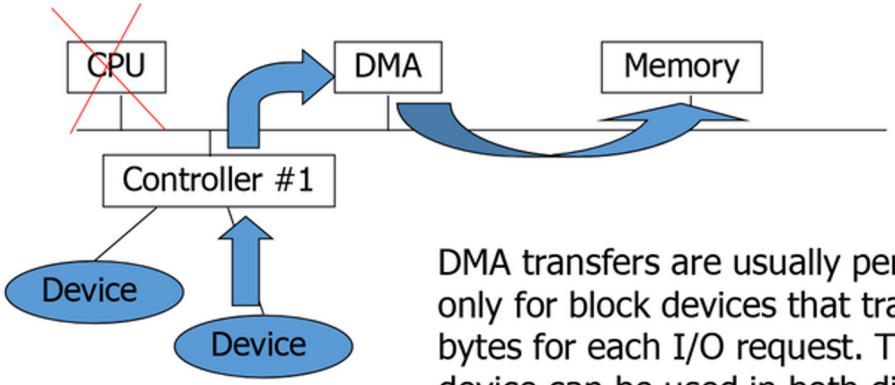
- Special hardware instruction used
- Atomically restores
  - Old program state
  - Interrupt mask
  - Instruction pointer (program counter)
- Hardware returns to location where interrupt occurred, and processing continues exactly as if no interrupt happened

Transfer Size and Interrupts

- Interrupt occurs after I /O operation completes
- Transfer size depends on device
  - Serial device transfers one character
  - Disk controller transfers one block (e.g., 512 bytes)
  - Network interface transfers one packet
- Large transfers use *Direct Memory Access* (*DMA*)
  - DMA devices manage data transfers between the controller and the system's primary memory independent of the CPU

Direct Memory Access (DMA)

- Hardware mechanism
- I /O device transfers large block of data to / from memory without using the processor
- Example: network interface places incoming network packet in memory buffer
- Advantage: allows processor to execute during I /O transfer
- Disadvantages
  - More expensive
  - More complex to design and program

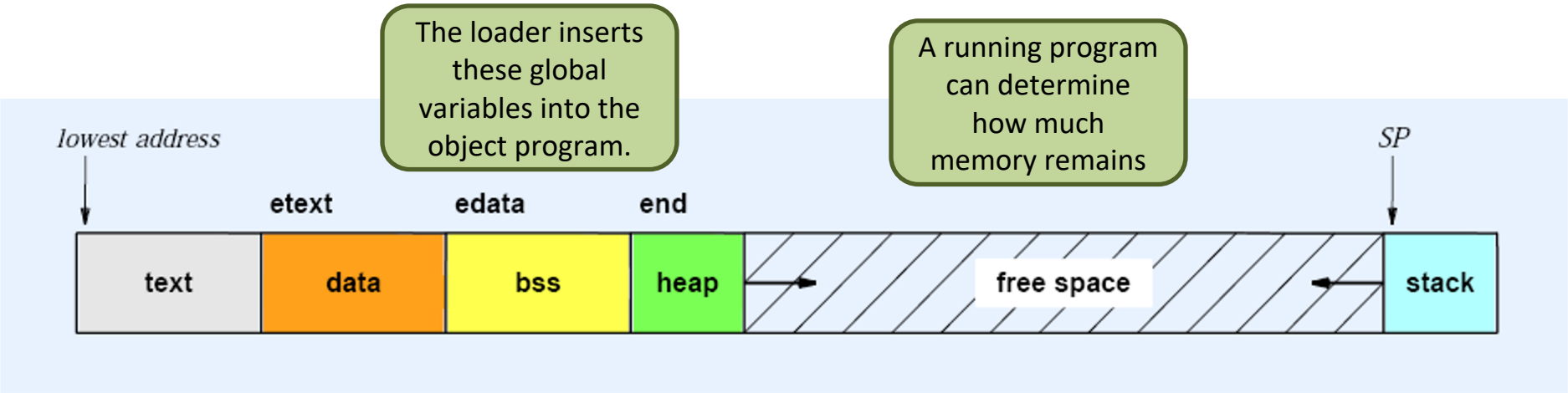


DMA transfers are usually performed only for block devices that transfer many bytes for each I/O request. The DMA device can be used in both directions.

Recap: Memory Segments in C Programs

- C Program has four primary data areas called *segments*
- Text segment
  - Contains program code
  - Usually unwritable
- Data segment
  - Contains initialized data values (globals)
- BSS segment
  - Contains uninitialized data values (set to zero)
- Stack segment
  - Used for function calls

Storage Layout for a C Program



- Stack grows downward (toward lower memory addresses)
- Heap grows upward

Symbols for Segment Addresses

- C compiler and/or linker adds three reserved names to symbol table:
  - _*etext* lies beyond text segment
  - _*edata* lies beyond data segment
  - _*end* lies beyond bss segment
- Only the addresses are significant; values are irrelevant
- Program can use the addresses of the reserved symbol to determine the size of segments
- Note: names are declared to be *extern* without the underscore:

```
extern    int    end;
```

Runtime Storage for a Process

- Text is shared
- Stack cannot be shared
- Data area *may* be shared
- Exact details depend on address space model OS offers

Example Runtime Storage Model: Xinu

- Single, shared copy of
  - Text segment
  - Data segment
  - bss segment
- One stack segment per process
  - Allocated when process created
- Each process has its own stack pointer

Summary

- Components of third generation computer
  - Processor
  - Main memory
  - I /O Devices
    - Accessed over bus
    - Operate concurrently with processor
    - Can be memory-mapped or port-mapped
    - Can use DMA
    - Employ interrupts

Summary
(continued)

- Interrupt mechanism
  - Informs processor when I /O completes
  - Permits asynchronous device operation
- C uses four memory areas: text, data, bss, and stack segments
- Multiple concurrent computations
  - Can share text, data, and bss
  - Cannot share stack