Big Ideas: First Half

- Static and dynamic
  - anything that can be determined before execution (by compiler) is called static
  - anything that can only be determined during execution (at runtime) is called dynamic

SM-213 Instruction Set Architecture

- hardware context is CPU and main memory with fetch/execute loop

Static Variable Access (static arrays)

- Key observations
  - address of b[a] cannot be computed statically by compiler
  - address can be computed dynamically from base and index stored in registers

Array access: use load/store indexed instruction

Learning Goals 1

- Memory
  - Endianness and memory-address alignment
- Globals
- Pointers
  - Pointers are C, G, and P operators, and pointer arithmetic
- Instance Variables
  - Instance variables of objects and structs
- Dynamic Storage
  - Dynamic storage allocation and deallocation
- If and Loop
  - If statements and loops
- Procedures
  - Procedures, call, return, stack, local variables and arguments
- Dynamic Flow Control
  - Dynamic flow control, polymorphism, and switch statements

Memory Access

- Memory is
  - an array of bytes, indexed by byte address
- Memory access is
  - restricted to a transfer between registers and memory
  - the ALU is thus unchanged, it still takes operands from registers

Common mistakes

- wrong: trying to have instruction read from memory and do computation all at once
  - instead should load memory into register as fast case, then do ALU operations on register contents only
- wrong: trying to have instruction do computation and store into memory all at once
  - won't compile, since write to memory is itself a write

Load into register

- variable value: direct memory value
  - from memory: base is register, offset as 4-bit number

Common mistakes

- wrong: interpreting 0x20 as 0
  - interpreting first character as a base 16 number
- wrong: converting 0x20 into format you don't particularly need
  - wasting your time converting into format you don't particularly need

Static Variable Access (static arrays)

- Key observations
  - address of b[a] cannot be computed statically by compiler
  - address can be computed dynamically from base and index stored in registers

Array access: use load/store indexed instruction

Learning Goals 2

- Read Assembly
  - Read assembly code
- Write Assembly
  - Write assembly code
- ISA-PL Connection
  - Connection between ISA and high-level programming language
- Asynchrony
  - PIC, DMA, interrupts and asynchronous programming
- Threads
  - Using and implementing threads
- Synchronization
  - Using and implementing spinlocks, monitors, condition variables and semaphores
- Virtual Memory
  - Virtual memory translation and implementation tradeoffs

Loading and Storing

- load into register
  - variable value: direct memory value
  - from memory: base is register, offset as 4-bit number

Common mistakes

- wrong: interpreting 0x20 as 0
  - interpreting first character as a base 16 number
- wrong: converting 0x20 into format you don't particularly need
  - wasting your time converting into format you don't particularly need

Not Covered on Final

- Details of memory management
  - Java weak references, reference objects, reference queues
  - slides 20-24 of module 1c, details of Lab 3 java memory leak solution
- C reference counting
  - slides 17-18 of module 1c
- Details of Hoare blocking signal for condition variables
  - slides 24-26 of module 2c
- OS/Encapsulation
  - module 2f
- Interprocess Communication, Networking, Protocols
  - module 2f

Numbers

- Hex vs. decimal vs. binary
  - In SM-213 assembly
  - Do in front of number means it's in hex otherwise it's decimal
  - converting from hex to decimal
    - convert each hex digit separately to decimal
      - base 16: 0x10 = 16, 0x1a = 10+6 = 16
  - converting from hex to binary
    - convert each hex digit separately to binary: 4 bits in one hex digit
      - convert each 4-bit block to hex digit

Alignment

- Power-of-two aligned addresses simplify hardware
  - required on many machines, faster on all machines
  - computing alignment: for what size integers is address X aligned?
  - byte address to integer address is division by power of two, which is just shifting bits
    - convert address to decimal: divide by 2, 4, 8, 16, .... stop as soon as there's a remainder
    - convert address to binary: away from right to left, stop when find a 1

Basic ALU Operations

- Arithmetic

Dereferencing Registers

- no dereference when you need it
  - extra dereference when you don't need it
  - example

Basic ALU Operations

- Arithmetic

Dereferencing Registers

- no dereference when you need it
  - extra dereference when you don't need it
  - example

Shifting, NOP and Halt

- Arithmetic

Not Covered on Final

- Details of memory management
  - Java weak references, reference objects, reference queues
  - slides 20-24 of module 1c, details of Lab 3 java memory leak solution
- C reference counting
  - slides 17-18 of module 1c
- Details of Hoare blocking signal for condition variables
  - slides 24-26 of module 2c
- OS/Encapsulation
  - module 2f
- Interprocess Communication, Networking, Protocols
  - module 2f
Implementing a Blocking Lock

- Spinlock guard
  - on for critical sections
  - off before thread blocks

```c
struct blocking_lock {
    spinlock lock; /* Spinlock guard */
    int waiter_count; /* # of waiting threads */
    thread_queue* waiter_queue; /* Waiting threads */
}

void lock(struct blocking_lock* l) {
    spinlock_lock(&l->lock);
    l->waiter_count = 0;
    l->waiter_queue = NULL;
}

void unlock(struct blocking_lock* l) {
    spinlock_unlock(&l->lock);
    waiter_thread->state = TS_RUNNABLE;
    ready_queue_enqueue(waiter_thread);
}
```

Blocking Lock Example Scenario

```c
void thread_a() {
    lock(&l);
    spinlock_guard();
    critical_section();
    spinlock_unlock();
    unlock(&l);
}

void thread_b() {
    lock(&l);
    spinlock_guard();
    critical_section();
    spinlock_unlock();
    unlock(&l);
}
```

Busywaiting vs Blocking

- Using spinlocks to busywait for long time wastes CPU cycles
- Use for short things
  - including write implementation of blocking locks
- Use for long things

Condition Variables

- Mechanism to transfer control back and forth between threads
- Uses monitors: CV can only be accessed when monitor lock is held
- Priority inversion: thread blocks until a subsequent notify operation on the variable
- Notify unlocks one waiter, continues to hold monitor
- Notify_all unlocks all waiters (broadcasts), continues to hold monitor
- Each CV associated with a monitor
- Multiple CVs can be associated with same monitor
- Independent conditions, but guarded by same mutex lock

Locks and Loops

- Confusion about spinlocks inside blocking locks
- Confusion about when spinlocks needed
- Must turn on to guard access to shared variables
- Must turn off before finishing or blocking
- Confusion about loop function
- Busywait only inside spinlock
- Block thread inside loop body, not busywaiting

Wait and Notify Semantics

- Monitor automatically re-entered before return from wait
- Monitor automatically re-entered before return from notify

Condition Variables

- Final will not cover Hoare blocking signal semantics
  - just non-blocking notify Hansen semantics

Common mistake:
- CVs do not have internal storage variables (boolean flags or int counters)
- CVs are variables named as such to protect from buffer overflows

Virtual Memory

- Virtual Address Space
  - An abstraction of the physical address space of main (i.e., physical memory)
  - Programs access memory using virtual addresses
  - Memory management unit translates virtual address to physical memory
  - MMU hardware performs translation on every memory access by program

- Process
  - Program execution with a virtual address space
    - May have one or many threads
  - Private address space required for static address allocation and isolation

Address Space Translation Tradeoffs

- Single, variable-size, non-expandable segment
  - Internal fragmentation of segment due to sparse address use
- Multiple, variable-size, non-expandable segments
  - External fragmentation of segments when size is not known statically
  - Moving segments would resolve fragmentation, but moving is costly

Deadlock and Starvation

- Solved problem: race conditions
  - Solved by synchronization abstractions: locks, monitors, semaphores
- Unsolvable problems when using multiple locks
  - Monitor accidentally re-enters when return from wait and notify
  - May cause deadlock

Translation: Search vs. Lookup Table

- Translate by searching through all segments: too slow!
- Example: search all segments for address
- Lookup table: faster, but requires additional memory

Translation with indexed lookup: Page Table

- Translate with indexed lookup: Page Table
  - For each segment, a table of page tables
  - Page table for each segment
  - Lookup by address
  - Paging structure
Demand Paging

- Key idea
  - some application data is not in memory
  - transfer from disk to memory, only when needed

- Page table
  - only stores entries for pages that are in memory
  - pages that are on disk are marked invalid
  - access to non-resident page causes a page-fault interrupt

- Memory map
  - a second data structure managed by the OS
  - divides virtual address space into regions, each mapped to a file
  - page-fault interrupt handler checks to see if faulted page is mapped
  - if so, gets page from disk, update Page Table and restart faulted instruction

- Page replacement
  - pages can now be removed from memory, transparent to program
  - a replacement algorithm chooses which pages should be resident and swaps out others

- Context Switch
  - Context switch: switching between threads from different processes
  - each process has its own page table
  - thread switch (save regs, switch stacks, restore regs)
  - page table switch
    - change PTBR (page table base register) so points to new page table
    - invalidate stale page table cache entries, may require flushing entire cache
    - page table cache: TLB (translation lookaside buffer)
    - fast cache storing recent page table translations
  - new process has no valid TLB entries, so many misses
  - many pages may need reloading from disk because of demand paging
  - thus context switch can be much more expensive than thread switch

Interprocess Communication

- Communication for processes that don’t share memory
  - on same processor or different ones connected by network

- Key ideas
  - client/server model, packet-based transport
  - naming endpoints: IP address and port
  - communication protocol layers
    - transport (TCP/UDP), routing (IP), data (Ethernet), physical (radio/cable)

- Sockets: OS abstraction for asynchronous control transfer
  - send: initiate sending message payload to receiving process, but do not wait
  - recv: receive next available message, either blocking or not if no data waiting

- Module not covered on final exam

Paging Summary

- Paging
  - a way to implement address space translation
  - divide virtual address space into small, fixed-sized virtual page frames
  - page table stores base physical address of every virtual page frame
  - page table is indexed by virtual page frame number
  - some virtual page frames have no physical page mapping
  - some of these get data on demand from disk

OS & Hardware Enforced Encapsulation

- Protecting operating system (OS) functions from application-level access
  - VM already protects memory: data in one address space cannot be named by process with another virtual address space
  - add hardware protection for OS function access

- User mode vs. kernel mode
  - all OS code/data included in every application page table and address space
  - split address space into two protection domains
    - application/user: checking VM to IP translation disallows access to OS part of space
    - user/kernel: everything accessible, including all system functionality
  - add user/kernel mode bit to each page table entry
  - add kernel mode register to CPU
  - protect switch from user to kernel mode: only through system calls
    - handled by interrupts with jump table in kernel memory

- Module not covered on final exam