- N bits → represent 2^N things
- doesn't always have to be value
- converting between bases, think buckets filled left → right
- 4 bits = nibble; 8 bits = byte
- Signed Decimal Representation
  - sign and magnitude - never used, 2 0's
  - 2's complement
    - $(-x) = \sim x + 1$ // to make negative
    - msb is $-(2^N)$
    - sign extension pretty easy to make more bits
    - 1 zero only, a little mean, widely used today
  - 1's complement
    - $-x = \sim x$ // just flip bits to make negative
    - max value is half. ex. 0111 → 1000
    - 2 0's, not really anymore
  - Bias Encoding
    - # = bits rep (unsigned) + bias
    - 00...00 most negative, 11...11 most positive

- C
  - string - array of chars, null terminated '\0'
    - malloc(strlen()+1) // for null terminator
    - int strlen(char* c) // length of c, until '\0'
    - int strcmp(char* s1, char* s2) // 0 if same
    - char* strcpy(char* dst, char* src) // src ⇒ dst
      - returns pointer to dst string
  - pointers - size of an unsigned int basically, need type for size of
    - int *a, b, c; // make 1 ptr, 2 vars
    - add integer, subtract, compare, compare to NULL, nothing else
      - *p++ == *(p++)
      - C knows size of pointers, so can increment right amount
    - casting always fine
  - array - n length, index 0, 1, ..., n-1
    - C defines 1 element past end of array must be valid, to check for end of array, compare to that extra address
    - interchangeable w/ pointers, not assignable
      - but array name is read-only pointer to start of array
  - struct - data structure made of smaller ones, need ; at end
    - can declare vars at end of struct declaration if want
    - '.' accessor
    - pointer to struct - use → dereference operator
      - p->x == (*p).x;
  - typedef - typedef <name> <alias>;
    - good for structs - typedef struct <name> {...} <alias>;
    - good for hiding pointers - typedef char* string;
  - Bus Error - anomalous condition on bus,
  - Segmentation Fault - access memory not allocated to it
  - a=b is assignment (returns a) vs. a==b
  - switch(<var>) { case <>: break; ...}
    - if no break; continues on
  - Storage Classes
    - auto, register - never used
    - extern - this defined elsewhere, default to 0
      - declaration vs. definition
        - functions must be declared before main()
      - int x=10; // when outside func, loaded before program "starts"
      - int x; // when outside function, is definition, can have multiple
      - extern int x; // uses x which is defined elsewhere
    - static - default to 0
      - scope - where var available, external vs internal linkage
      - extent - how long var lasts
      - inside procedure - local scope, infinite extent
      - outside procedure - same-file scope, infinite extent
      - function - ↗
      - vars initialized once, value persists between calls

- int main(int argc, char* argv);
  - can also do char* argv[]
  - argv[0] is program name
- little endian - smallest byte on right
  - big endian - big byte right
- #define <stuff> copy and paste
  - preprocessor replace directly
- getchar() and putchar()
  - int getchar() to extra characters
  - != before =
- i++ vs. ++i
- && before ||, left to right
- if, else if, else
- local var - 'auto' be fixed by default
  - garbage initially
- extern
  - access by name by any func
  - defined once, declared everywhere
  - usually place at beginning of file
  - declaration - stated, no storage
  - definition - stated, storage allocated
  - by default functions have extern
  - extern int x=10; can't do a definition also
- variable names - begin w/ letter
  - letter, _, digits no '-'
- constants - 10L decimal L for long, U for unsigned, leading 0 for octal
  - default double of decimal point, F for float, L for long double
- enum - allows int to be define
  - very similar to struct
  - first value is 0 unless otherwise stated
  - enum <name> {...} <vars>;
- arithmetic operators = +, -, /, %, *
- relational operators >, >=, <=, <
  - == and != before in precedence
  - all lower than arithmetic operator
- &&, || short-circuit
- bitwise operators - &, |, ^, <<, >>, ~
  - higher than arithmetic
- Control Flow
  - don't care about indent
  - break; // exit loop
  - continue; // back to top
  - do {} while();
  - goto - avoid
- static - limit scope to source file
  - can only init w/ constant literals
- array - if use {}, sets extra to 0
- printf: %d - decimal; %u - unsigned
- void* - exists!
- pointer to function
  - void (*func)(int, int) = <name>
- array don't know own length, # are meaningless
- handle - pointer to pointer - address of a pointer
- always free() or malloc()!
- size of() in bytes, knows size of array
- malloc() // need a NULL check
- ternary operator - <cond> ? <if>: <else>
- const - don't change it
- should cast malloc()

- C Memory Management
  - Aside
    - structure declaration don't allocate memory
    - once variable declaration allocates it
  - char* c = "hi"; // static
  - char c[] = "hi"; // stack

  - code - loaded when program starts, doesn't change
  - static - var declared outside main, does not grow or shrink; global/static
    - global variable stays permanent (compiler time constants only)
  - heap - space requested for pointers, resize dynamically, grows upward
    - dynamic malloc storage, data live until deallocated
  - stack - local vars, grows down, include main()
    - local vars
- Stack Frame - function calls LIFO, must return address from here
  - stack pointer tells where top stack frame is

- Heap
  - malloc() calls - only one granularity contiguous blocks, sporadic can't spread
  - want to avoid fragmentation
  - want minimal memory overhead
  - stored as circular linked list - size of block, pointer to next block
  - choosing a block
    - best fit - smallest block big enough, results in slivers
    - first fit - first block big enough, many small blocks at beginning
    - next fit - first fit but remember where finished searching
      - doesn't let slivers at beginning, better than first-fit

- IEC Prefixes
  - Ki: - Kibi: - 2^{10}
  - Mi: - Mebi: - 2^{20} — increases by power of 10
  - Gi, Ti:, Pi, Ei:, Zi:, Yi:
  - gibi, tebi:, pebi:, exbi:, zebi:, yobi:
- SI
  - K, M, G, T, P, E, Z, Y
  - Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta
- ½ bit / 0 bit resp / value

- Floating Point
  - Fixed Point - used "binary" point, math: $a \cdot 10^b$; a-mantissa, b-exponent
  - Format $1.xxx\ldots x * 2^{yyy\ldots y}$, normalized with 1 on left
  - 32 bit float (fp32) - 1 bit sign, 8 bit exponent, 23 bit significand
    - $\pm 1.0 \cdot 10^{-38}$ to $\pm 2.10^{38}$ (sci is negative)
  - 2 zeros tho
  - overflow - exponent too big
  - underflow - number too small
  - counting 0 → max → −0 → −max
  - IEEE 754, bias of −127 for exponent
    - $(-1)^S * (1 + \text{significand}) * 2^{\text{exponent}-127}$
    - special cases
      - all zero = 0, sign bit can make −0
      - ∞ ; biggest exponent, significand all 0
      - NaN; biggest exponent, non 0 significand
      - Denorm - 0 exponent, 0.xxx, # = 0.xxx * $2^{\text{denormexponent}}$ ← −126 for zero-ish
        - allows # < min
    - $2^{24}+1$ is smallest positive integer can't represent
  - adding tiny num to big num just returns big number b/c of precision & careful

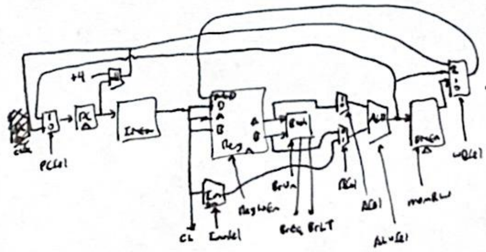
Matthew Tran
CS61C

cs61c-azo

- RISC-V is an assembly language
  - RISC - let software do the rest
  - open source license free spec
  - 32 registers x0 ... x31
    - 32 bit each, x0 == 0 always
  - Assembly
    - each line 1 instruction, # comments
    - memory address
      - 4 bytes = 1 word
      - little endian - smallest byte in smallest addr
    - lw rd, imm(rs1) # rd = MEM[rs1 + imm]
    - sw rs2, imm(rs1) # MEM[rs1 + imm] = rs2
    - lb, sb equivalents, remember little endian, sign extended
      - lbu - unsigned, 0 extend
    - bge rs1, rs2, label # goto label if rs1 > rs2
    - jal rd, label # rd = PC+4, goto label → PC = PC + imm
    - jalr rd, rs1 # rd = PC+4, goto rs1 addr, PC = rs1 + imm
    - sll, srl, sra - sra sign extends, all have immediate versions
    - and, or, xor; not → xor w/ FF...
    - pseudoinstructions - j, li, la, mv, nop, not, ret
    - Call Convention
      - keep eye on who saves what, use Green Card
      - if calling function, make sure to store ra
      - use sp, addi sp, sp, -4 to reserve 4 bytes
    - Instruction Formats
      - R - register-register ops
        - opcode - distinguishes operation, partially
        - funct7 + funct3 → fully specifies
      - I - register immediate ops
        - imm is 12-bit signed, extended to 32 bit in hardware
        - -2048 to 2047
        - shift by imm ignores the upper bits
      - S - stores aka SB
      - B - branches
        - imm -4096 to 4094 as 2 byte offsets
        - 2 not 4 bc compressed instruction set
        - drop the 0 bit bc don't need it
      - U - upper immediate stuff, for huge imms
        - imm [31:12], upper 20 bits
        - lui rd, imm
          - clear lower 12 bits, store imm
          - use addi to set lower 12 bits
          - but addi does sign extension so need to preadd 1
          - li handles all of this
        - auipc - rd = PC + imm
          - good for storing PC
      - J - jumps, aka UJ
        - like B, drops 0 bit, $\pm 2^{19}$ 2 byte locs, $\pm 2^{18}$ instr
        - jump to anywhere
          - auipc x1, <hi 20 bits>
          - jalr x0, x1, <low 12 bits> # careful sign extension

- CALL
  - Compiler
    - inputs: high-level language
    - output: assembly language, may have pseudoinstructions
  - Assembler
    - inputs: assembly, foo.s
    - outputs: object code, foo.o, information tables (true assembly only)
    - Assembler Directives
      - .text - assembly
      - .data - put in user data segment (static)
      - .global sym - let sym access from other files
      - .string str - store str and null terminate
      - .word w1...wn - store n 32 bit words in successive locs
    - Once no more pseudo-instructions - can find actual jump distance
    - "Forward reference" problem - need two passes to find everything
    - PIC - position independent code
    - what about static data? make tables to tell linker
      - Symbol Table - items used by other files, ex labels, .data
      - Relocation Table - stuff this file needs, like exact locations
        - ex. external labels, static data
    - object File Format
      - object file header - size/pos of other pieces
      - text segment - machine code
      - data segment - binary rep of static data
      - relocation info - lines that need fixing
      - symbol table - list of labels and static data that can be ref.
      - debugging info
      - usually ELF file format

- Synchronous Digital Systems (SDS)
  - clock - we use rising edge
  - D-type flip flop
    - D is "data", Q is "output"
    - Q becomes D on rising edge
    - t_{setup} - time stable before clock pulse
    - t_{hold} - how long after clock pulse stable
    - $t_{clk to q}$ - time for Q to become D
    - undefined if violate anything
  - max delay = $t_{setup} + t_{clk to q}$ + longest$_{CL}$
    - determines max clock speed
  - max hold = $t_{clk to q}$ + shortest$_{CL}$
    - determines tightest logic loop
- FSM (Finite State Machines)
  - finite # of states
  - use PS (present state) and input to determine next state (NS)
  - use truth table
  - usually in/out →
- Boolean Algebra (·, +, ¯)
  - Complementarity - $x \cdot \bar{x} = 0$ $x + \bar{x} = 1$
  - laws of 0s and 1s - $x \cdot 0 = 0$ $x + 1 = 1$
  - identities - $x \cdot 1 = x$ $x + 0 = x$
  - idempotent - $x \cdot x = x$ $x + x = x$
  - commutative - $x \cdot y = y \cdot x$ $x + y = y + x$
  - associative - $(xy)z = x(yz)$ $(x+y)+z = x+(y+z)$
  - distribution - $x(y+z) = xy + xz$ $x + yz = (x+y)(x+z)$
  - uniting theorem - $xy + x = x$ $(x+y)x = x$
  - Demorgan's - $\overline{x \cdot y} = \bar{x} + \bar{y}$ $\overline{(x+y)} = \bar{x} \cdot \bar{y}$
    - $x + \bar{x}y = x + y$
  - ⊐D and, ⊐D or, ⊐⊐D xor, ▷o not



- Single Cycle Datapath and Control
  - ALU (Arithmetic Logic Unit)
    - adder
      - half adder - sum - xor, carry - and
      - full adder - sum - xor, carry - majority
    - overflow?
      - unsigned - highest c_out
      - signed - $c_n \wedge c_{n-1}$, out of highest bits
    - sub - xor and put 1 in carry in, basically 2s complement
  - Stages
    - IF - instruction fetch 200ps
    - ID - instruction decode 100ps regread
    - EX - execute (ALU) 200ps
    - MEM - memory access DMEM 200ps
    - WB - write back regwrite 100ps

    Typ speed 1.25 GHz
    bc of lw uses all of them
  - RISC-V has 47 instructions, 37 enough for any C code
  - RTL (Register Transfer Level) - describe instructions as C, Verilog
  - Control Logic - use * when don't care, simplify
    - ROM - popular, easily programmable, fix errors
    - CL - use gates, convert truth tables
    - BrEq - if equal
    - BrLT - less than
    - BrUn - branch unsigned
    - PCSel - ALU or PC+4
    - ImmSel - which immed format
    - ASel - Reg or PC
    - BSel - Reg or Imm
    - ALUSel - add/or/sub
    - MemRW - write to DMEM
    - RegWEn - write to register
    - WBSel - what to write
  - Iron Law of Processor Performance
    - $\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$ (Cycles/Instruction ← CPI)
    - $\frac{\text{Energy}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Energy}}{\text{Instruction}}$ (↑ CV^2)
  - Energy "Iron Law" -
    - $\frac{\text{Tasks}}{\text{Second}} = \text{Power}\left(\frac{J}{s}\right) \cdot \text{efficiency}\left(\frac{\text{Tasks}}{\text{Joule}}\right)$

- Linker
  - inputs: object files, info tables
  - outputs: executable code
  - "links" several .o files, allows separate compilation
  - steps
    - put all .text together
    - put data together, add to end of text
    - resolve references - use relocation table
      - fill in absolute addresses
  - 4 address types
    - PC-relative (beq, bne, jal; auipc/addi)
      - PIC, no need relocate
    - absolute function addr (auipc/jalr) } always relocate
    - external function ref (auipc/jalr) } always relocate
    - static data ref (lui/addi)
      - loads and stores to static, relative to global pointer (gp)
  - Resolving References
    - assumes first word of text at 0x10000 for RV32
    - search in "user" symbol tables
      - not found → search library files
      - fill in machine code appropriately
  - Static vs Dynamically Linked Library
    - DLL allows don't have to link new copy w/ compilation
    - less space, less time sending, less memory if shared
    - but more time overhead to do link
- Loader
  - load executable into memory and run, usually OS job
  - bunch of stuff
  - reserve enough memory

Matthew Tran
CS61C
pg. 3
cs61c-a20

- Pipelining
  - Add registers between stages
  - shortest clock period now max of stages
  - can't get 5x speedup bc reasons
  - Hazards
    - Structural
      - resource busy, needed in multiple stages
      - solutions
        1) stall
        2) add more hardware (always ok)
          - RegFile:
            - access both in ID and WB
            - sol: have separate read and write ports
            - double pumping - for same register, write in 1st half, read in 2nd half
            - asynchronous read/write
        - Memory
          - IF and MEM same time
          - sol: have separate DMEM and IMEM
          - actually just use 2 caches
    - Data
      - prev instr hasn't finished writing yet
      - stalling - add NOP ← compiler optimization can insert other instr
      - forwarding - use the ALU instead of register
        - good when same stage
        - lw still have to stall
    - Control
      - caused by jump and branch
      - could stall, but slow
      - use branch prediction
        - if wrong, flush the pipeline, insert nops
- Other Performance Increase
  - clock rate - limited by tech and power dissipation
  - pipelining - good, but hazards, CPI > 1
  - Superscalar Processor
    - instruction level parallelism
    - not multicore, but replicate pipelines
    - start multiple instructions per cycle
    - CPI < 1
    - "out of order" execution - reduce hazards
- Cache
  - DRAM slow, use SRAM close to chip, smaller
  - what to store?
    - Temporal locality - used it before, prob need it again
    - Spatial locality - prob need neighbors
  - Direct Mapped Cache
    - each address mapped to one block in cache
      - block - unit of transfer between cache and memory
      - bigger block = better spatial locality + more miss penalty
  - TIO - each address split ← bigger if → ← smaller if more associative
    - Tag: like key, ID's each block
    - Index: which row of cache to check, which block
    - Offset: specific byte within a block
  - Access Read
    - processor checks cache
    - cache checks
      - hit: ✓ send value to processor
      - miss: X get from memory, replace, send new value
  - valid bit - says if something there
  - Cache writing
    - write-through - update both, slow
    - write-back - dirty bit, write when need to
  - Cache Misses "Three C's"
    - Compulsory - can't have bc never asked for it
    - ~~Conflict~~ Capacity - had before, but got replaced bc no room
    - Conflict - index conflict, got overwritten
  - 4th C - coherence - core conflict, pretty big one
  - Cache Structure
    - Direct-mapped - one block per index #sets = #blocks
    - N-way set associative - block in any one of N places #blocks/N sets
    - Fully associative - #blocks compares to check if in cache
  - Block Replacement Policy
    - for cache w/ associativity
    - LRU (Least Recently Used) - kick least used one out
    - FIFO - queue
    - Random - good for low temporal locality
  - Average Memory Access Time (AMAT)
    - AMAT = hit time + miss rate * miss penalty
      - cache checked on both hits and misses
    - multilevel caches to decrease this

- Virtual Memory "infinite"
  - give each process illusion of own "memory"
  - provides protection/security
  - all programs start at 0
  - memory split into "pages"
  - each process has own page table in memory
    - Page Table Base Register (PTBR) - holds page table's address
    - when OS switches process, save page table data to disk, then swap in new page table ← context switching
      - invalidate all TLB
  - Terms
    - Virtual Address (VA)
    - Virtual Memory (VM)
    - Virtual Page Number (VPN)
    - Physical Address (PA)
    - Physical Memory (PM)
    - Physical Page Number (PPN)
  - split VA = VPN | offset
    PA = PPN | offset
  - Page Table Entry (PTE) - valid bit, access rights, PPN
  - Translation Lookaside Buffer (TLB) ← also valid bit, access rights for
    - cache for the page table
    - usually highly associative / full
    - VPN split into tag and index (no index if fully assoc)
  - Memory Read
    - check TLB
      - TLB Hit ✓ get PPN
      - TLB Miss - check page table (PT)
        - PT Hit - load PTE into TLB, acts as a cache miss
        - PT Miss - aka Page Fault, fetch disk to memory, update PTE, load to TLB
          - expensive, but < 0.001%
          - create a new mapping ← LRU
          - eviction - memory full, kick one out
          - no eviction - make new mapping
    - check cache w/ PA
  - protection fault - wrong access bits
  - Thrashing - constant disk memory swap
- Parallelism
  - Flynn's Taxonomy
    - SISD - single instruction, single data stream
    - SIMD
    - MISD - not used
    - MIMD
    - SPMD - single program multiple data
  - SIMD
    - Data Level Parallelism (DLP)
    - MMX, SSE, AVX
  - C Intrinsics - assembly in C
  - Multiprocessor Execution Model
    - each core runs its own instructions
    - separate and shared resources
  - Threads
    - list of instructions for one task
    - has PC, registers, access to shared memory
    - each physical core has 1+ threads
    - OS multiplexes them
      - multiplex software threads onto hardware threads
        - switch out blocked threads (cache miss, user input, network access)
        - switch on a timer
      - remove software thread by interrupt, save their PC and registers
      - load one by load registers + PC, jump to it
    - (- can share / update memory, not code ← the stack)
  - Multithreading
    - must perform switch really quickly
    - Hardware Assisted Software Multithreading
      - Hyperthreading - both threads run simultaneously SMT
        - 2 copies of PC and registers
        - 5% more hardware, ~2x performance
  - Logical Threads
    - pretty much what can handle a software thread
  - OpenMP
    - C library handle parallelism
    - be careful of data dependencies !
      #pragma omp parallel // runs code multiple times in each thread
      omp_get_num_threads()
      omp_get_thread_num()
      #pragma omp parallel for // handles for loop split, must be in proper mode;
      #pragma omp critical // only one thread at a time
      #pragma omp reduction(<op>: <var>) // handles private var and all that
      // can combine them
  - MapReduce and Spark - map function, then reduce; trivially parallelized
    - Resilient Distributed Datasets (RDD)
    - map(f), flatMap(f) ← each input 0+ items, reduceByKey(f), reduce(f)

- Parallelism (cont.)
  - only one person at time access shared resource
    - must take turns
  - Locks
    - variable that says being used
    - issue is check and use are 2 instructions
      - possible 2 processes access same time
    - solution:
  - Hardware Synchronization
    - atomic read/write - 1 instr
    - must use shared memory
    - usually swap register and memory
    - ...
    - RISC-V Atomic Memory Operations (AMOs)
  - Deadlock - everything depends on each other, cycle
    - soln? - measure elapsed time
  - Multicore Multiprocessor
    - (Shared Memory) Symmetric Multiprocessor (SMP)
      - all multicore cpus today are SMP
      - 2+ identical cores
      - single shared memory (each one still have own cache)
    - communicate via a shared variable
  - Multiprocessor Cache
    - How keep coherency?
      - if miss/write, tell others, invalidate copies
    - MOESI - cache block states
      - Modified - up to date, memory wrong, no one has copy
      - Owner - up to date, others have copy (in shared state)
        - doesn't need to write to memory (yet), lets others read
      - Exclusive - up to date, no cache copy, memory up to date
        - avoids writes to memory, lets others read from it
      - Shared - up to date, other caches may have copy
      - Invalid - nothing
      (← variations)
    - owner - used shared to let others read owner instead
    - exclusive - ... modified to avoid write to memory on write
    - False sharing - watch block size, unnecessary reloading of data
      that is modified, i.e. f.x and f.y in same block
  - Amdahl's Law
    $speedup = \frac{1}{(1-p) + p/s}$ p: proportion that parallel s: nx speedup for parallel
  - Request Level Parallelism (RLP) - many independent requests
  - Data Level Parallelism (DLP) - data in memory / disks
  - Cloud Computing
    - Warehouse Scale Computers (WSC)
      - 10k to 100k computers
      - < 1 hour down/year 99.99%
      - cost of ownership >> cost of purchasing equipment
      - need high efficiency Power Usage Effectiveness (PUE) = $\frac{total\ power}{IT\ power}$ want 1.0
      - cooling big issue - air flow, regular ones, water, climate
      - also want maintain high CPU usage
- I/O
  - processor needs to input and output bytes to interact
  - can use special I/O instructions and hardware
  - or use Memory Mapped I/Os (MMIO) - very common
    - control register, data register
  - Polling
    - constantly ask, good for mouse, bad for HDD
  - Interrupts
    - device tell CPU, trap handler PC saved in Machine Exception PC (MEPC)
      - store registers, handle code, come back, reload
  - Interrupt - external, asynchronous
  - Exception - caused by program
  - Trap - act of "servicing" an interrupt or exception
  - Precise Traps - ensure every instr prior to trap done, none after done
    - don't need to know pipeline, just jumps back
    - handled like a pipeline hazard
      - 1) complete instr before trap
      - 2) flush curr instr's
      - 3) optionally store exception in status register
      - 4) transfer execution to trap handler
  - Golden Age of Computer Architecture
    - software advances can inspire architecture innovations
    - raising HW/SW interface ... opportunities for innovation
    - marketplace settles architecture debates
    - control vs. datapath
    - microprogramming - logic used to design control
    - Dennard scaling - power for chip same as ... increasing, no more
    - Security challenges
    - Domain Specific Architectures (DSA) - do few tasks well
    - Tensor Processing Unit (TPU) - matrix unit - multiply accumulate 65536 MAC 700MHz 8GB DDR3 4MB ... weight buffer
      - do more ops, > 25x MAC vs GPU, > 100x vs CPU
      - small ... memory ... off board memory
  - GPU
    - ... determine ...
    - many cores, TLP, software coherency, low freq, high latency
      many registers, limited branch prediction, in order, many execution