**323 Review**

**Chapter 1**

What evidence do you see of the ongoing "computer revolution"?

Computers in Cell Phones, in automobiles, Human Genome, Internet, Search Engines, even toothbrushes

What is "Moore's Law"?

Integrated circuit resources would double every 1-2 years (18 months), no longer accurate

What three application classes does our text divide computers into, and what are the key characteristics of each?

Personal Computers: High performance, low cost, execute 3rd party software

Servers: Accessed only via a network, small website runners to supercomputers that contain terabytes of memory

Embedded Computers: Largest class of computers, they are everywhere

How does a mebibyte differ from a megabyte?

Mebibyte = 1024 kibibytes (2^20 bytes), Megabyte = 1000 kilobytes (10^6 bytes)

What other new terms are introduced for common sizes, and what ambiguity do they resolve?

Kibibyte = KiB vs kilobyte = KB. ronnabyte = 10^27 and queccabyte = 10^30

What are the distinguishing characteristics of computing in the "post-PC era"?

PMDs (Personal Mobile device) Wireless connectivity to internet, can download apps, no keyboard and mouse, some touch screen or perhaps speech input

Can you define these terms?

personal mobile devices: Battery operated with Wireless connectivity to internet, can download apps, no keyboard and mouse, some touch screen or perhaps speech input

Cloud Computing: Relies on giant datacenter called Warehouse Scale Computers (WSC’s)

Software as a Service: (SaaS) Selling software to companies so that they don’t have to make their own. Companies build apps for other companies

What are the "eight great ideas in computer architecture" identified in our text?

Can you identify examples of each?

**Use abstraction to simplify Design**: Building self-driving cars whose control systems make use of existing sensor systems already in smart vehicles, such as lane departure systems and cruise control systems

**Make the Common Case Fast:** Express elevators in tall buildings

**Performance via Parallelism:** Increasing the gate area on a CMOS transistor to decrease its switching time

**Performance via Pipelining:** Assembly lines in automobile manufacturing

**Performance via Prediction:** Aircraft and marine navigation systems that incorporate wind forecasts

**Hierarchy of Memories:** Library reserve desk (where high-demand library materials are available for short-term checkout)

**Dependability via Redundancy:** Suspension bridge cables

**Designing for Moore’s Law** – now redundant – build smaller devices to prepare for smaller IC’s

What key roles does systems software play in modern computer systems?

Convert the software to hardware functions

What steps are involved in the translation of a program from a high-level language to a form the hardware can understand?

High Level Language Program -----Compiler-------

Assembly Language program ------Assembler-----

Add other relevant files —--- Linker—----

Binary Machine Language Program -- tells the hardware what to do

What advantages do high-level languages offer, relative to programming in assembly language or machine language?

High-level language: Easier to read, easier to code and consider logic

Assembly Language: possible to read, no compile time so faster

Machine Language: impossible to read, no compile or assembly time

What are the five classic components of a computer?

Input, output, memory, Datapath, and control (Datapath and control are the processor)

On typical systems, how does a processor change what is visible on a graphics display?

It uses a bit map (matrix of bits) to represent the pixels on the screen, and changes transistor switches to change the intensity of the red, green, and blue light

What kinds of memory are said to be volatile?

Memory that is not remembered when power is removed, EX. DRAM used to keep track of programs that are currently running

Which memory or storage technologies are nonvolatile?

Memory that is remembered even when power is removed, EX. DVD disk, storage

How does the primary memory differ from the secondary memory?

Primary Memory = “Main Memory” Volatile, DRAM

Secondary Memory Nonvolatile, storage, stuff that stays even when power is off

What is a local area network? A wide area network?

Local Area Network Short range (under a kilometer) connection, with switches that can also provide routing services and security

Wide Area Network Cross continents, the backbone of the internet

What are key steps in the manufacturing of integrated circuits?

Silicon Ingot Slicer, Blank Wafers, 20-40 steps Patterned wafers,

Wafer tester, Tested wafer Dicer, Tested Dies, Bond die to package, Packaged Dies Part tester, Tested Packaged Dies Ship to Customers

What is the best way to measure computer performance?

Execution time

How are performance and execution time related?

Performance(X) = 1/Execution time(X)

What precisely does it mean if "computer X is n times faster than computer Y"?

The execution time for computer X is n times shorter than computer Y

If Performance(X)/Performance(Y) = n = Exec time(Y)/ Exec time(X) then x is n times as fast as y

What is the difference between elapsed time and CPU execution time?

CPU Execution time = Time to complete task without counting time spent on I/O or other programs

Elapsed time = Total time to complete task

What components of CPU execution time might be separately tracked and reported?

CPU execution time for a program, Instruction count, Clock Cycles per instruction (CPI), Clock cycle time

What is the "classic CPU performance equation" and what aspects of the hardware and software affect each of its terms?

Exec time = (Instruction count \* CPI) / Clock rate

What was the "power wall" that computer designers ran into, and how did their designs change as a result?

Energy and power can be reduced by lowering voltage, to accommodate higher clock rates. But voltage can only be lowered so much before there starts to be leakage from transistors, just like a drain that can’t be shut all the way.

Their designs changed from Uniprocessors to Multiprocessors on chips

Additionally, we can no longer increase power because we’ve reached the **realistic** limits of microprocessor cooling technology. Increasing power any more would result in too much heat. We can go up to as high as 300 watts and keep things cool but it’s far too expensive.

What role do benchmark sets play in evaluating the performance of computer systems?

They are programs that are specifically chosen to measure performance. They are used as the “common case” in order to “make the common case fast.” They basically form a workload that the user hopes will predict the performance of the actual workload

What is Amdahl's Law, and why is it useful?

Basically, it is used for evaluating possible enhancements. We can estimate performance improvements if we know time consumed for some function and the potential speedup.

The central idea is that improving one part of a program can only speed up the program by so much.

**Chapter 2**

* What is the *instruction set* of a computer?

Instruction set = set of instructions the computer knows how to use. Humans usually code in high level languages or in assembly.

* + How are instructions typically represented for human use?

Instructions for humans are represented as high level programming language such as C or Java.

* + In what form are instructions when they are executed by the hardware?

The hardware executes binary ones and zeros or machine language.

* What is the *stored-program concept*, and why is it "the secret of computing"?

stored-program concept = The idea that instructions and data of many types can be stored in memory as numbers and thus be easy to change, leading to the stored-program computer. Without stored numbers as programs, every time a new program was made, it would have to be done from scratch starting from the basic machine code to the assembler all the way to the high-level language.

* What aspects of the RISC-V instruction set illustrate each of the three design principles described in Chapter 2?

1. Simplicity favors regularity
   1. Regularity motivates many features of the RISC-V instruction set: keeping all instructions a single size, always requiring register operands in arithmetic instructions, and keeping the register fields in the same place in all instruction formats.
2. Smaller is faster
   1. RISC-V only has 32 registers to keep things smaller.
3. Good design demands good compromises.
   1. All instructions are the same length to provide for large addresses and constants. Only one instruction per line is a compromise for good design.

* What are the consequences of doubling the number of registers in a computer?

Increasing their number increases the number of bits taken in the instruction format to indicate which register is being acted on.

Also, a very large number of registers may increase the clock cycle time simply because it takes electronic signals longer when they must travel farther. Pg. 73

* What data elements of a program are likely to be stored in memory rather than registers?

Memory Operands, //FIXME

* In RISC-V, how is a memory address determined for a load or store?

Based on the memToReg or MemWrite signals which is determined by the op code in the instruction. —> This tells whether it is a load or a store

Also important: memory address is the register in () + offset for a load or a store

This is talking about the memory address to load from or store to. In the instruction it will have an immediate and a base address. 40(x13) for example would start at the base address contained in register x13 and go forward 4 I’ll 0 bytes or 10 words. If x13 was the base address of an array A, 0(x13) would be A[0]. 40(x13) would be A[10].

* What is the difference in address values for adjacent elements of an array of doublewords?

8, for doublewords instead of the usual 4 for words

* What does it mean for data elements to be *aligned* in memory?

I’m pretty sure this is just how the stack, heap, and other memory are “aligned” for easy access

Data is aligned to fill up chunks, usually chunks of 4 bytes, or at least one byte. This technically wastes space, but increases read and write speed.

* + Does RISC-V have *alignment restrictions*?

I think yes, it does, since it restricts words to 32 bits. The standard is to only store and load every 4 bytes for a full word. We can also load a byte or a half-word. But we would not load or store a word at memory location 2, because it is not a multiple of 4.

* What does it mean for a program to *spill* registers?
  + The process of putting less frequently used variables (or those needed later) into memory
* What are the advantages of arithmetic instructions that have an *immediate* operand?
  + You can use a constant in an operation without needing to load one from memory.
  + In these instructions, where is the constant immediate value actually stored?
    - It is part of the instruction. It is stored in the designated ‘immediate’ bits of an instruction.
* In RISC-V, how does x0 differ from other registers?
  + The x0 register is ALWAYS 0 and cannot be updated to anything else.
* By convention, is bit 0 the *least* or *most* significant bit in a word or doubleword?
  + Least significant. [Wikipedia](https://en.wikipedia.org/wiki/Bit_numbering)
* What is sign extension?
  + Sign extension is a repetition of the most significant bit (MSB) when storing a smaller data size into a larger size.
  + How does the lb instruction differ from the lbu instruction?
    - Load Byte → R[rd] = {24b’M[ ](7), M[R[rs1] + imm](7:0)}
    - Load Byte Unsigned → R[rd] = {24’b0,M[R[rs1]+imm](7:0)}
    - These both take the least significant byte at the memory location and save it to rd. The difference is how those 8 bits are sign extended. Lb extends using bit 7. Lbu just extends with zeros.
* What is the difference between *assembly language* and *machine language*?
  + Assembly is readable simple instructions that directly represent binary numbers. Machine language is the pure binary numbers representing instructions.
* What fields exist in RISC-V instructions?
  + Opcode, funct7, funct3, rd, rs1,rs2, imm
  + Which fields are always in the same place?
  + Opcode, rd, rs1, rs2, func 3, func 7. These are always in the same place when they are present. The only field that isn't consistent is the immediate field.
  + How does the hardware know the format of any given instruction?
  + Opcode
* Why did the RISC-V designers not include a subi instruction?
  + Because you can addi a negative number. Simplicity.
* How does the sra instruction differ from the srl instruction?
  + Sra >>> shift right arithmetic (Sign Extension)
  + Srl >> shift right logical (no sign extension)
* How can logical operations be used to read and write arbitrary bit fields within a doubleword?
  + We can shift left and right. Also can and, or, xor with bitmasks to manipulate specific bit fields in a word.
* How are conditional branches in RISC-V typically used to implement if statements and loops?
  + If a condition is met an instruction can branch back up to the top of a ‘loop’ or can branch out of the ‘loop’.
* What is a *basic block*?
  + A sequence of instructions without branches in the body of the block. There may be a branch at the bottom or a branch target/label at the top
* How does the blt instruction differ from the bltu instruction?
  + Blt >> branch less than (branches if one number is less than another, can use negatives)
  + Bltu >> branch less than unsigned (branches if one number is less than another, unsigned numbers only, no negatives)
  + How can unsigned comparisons simplify out-of-bounds checks for array indices?
    - ??? Pg 102.
* How can a *branch address table* be used to implement a C switch statement efficiently?

If you were to code in Assembly, you would have a bunch of different branches based on the different case statements from C. Based on which one evaluated to true, it would go into that branch and perform the instructions there then probably jump and link back or just return somewhere after the branch statements.

* How are procedures implemented in RISC-V code?

Usually with branches or jump instructions.

* + How are parameter values passed to functions, and how do functions return values?

Function argument registers x12-17 are used to help parameter values and return values are stored into the return address register x1.

* + What instruction is used to call a function, and how is the return address saved?
  + We use a jal instruction to jump to a block and link the return address to x1 which is by convention ra
  + What instruction is used to return from a function?
  + Ret is the sudo instruction used to return. It actually uses jalr to jump to reg 1.
  + How is the stack used to support procedures, and under what conditions is a stack frame allocated?
  + As the stack grows down, the values below the stack include stack frames from earlier procedure calls, which include return addresses.
  + Memory is allocated on the stack by sp = sp - (how much you need). This is done when new functions are called inside other functions or in recursion.
  + Sp is saved in register 2 by convention.
* What is the *heap* and how is it used by C programs?
  + Dynamic data: heap, Memory used by malloc in C. It is allocated and does not change until it is unallocated.
* What memory-access instructions would you expect to see in a program that manipulates ASCII strings?

lbu/sb(sw)

* What does the lui instruction do?
  + Load upper immediate (lui) to load a 20-bit constant into bits 12 through 31 of a register. The rightmost 12 bits are filled with zeros. This instruction allows, for example, a 32-bit constant to be created with two instructions.
* What is *PC-relative addressing*, and which RISC-V instructions use it?

An addressing regime in which the address is the sum of the program counter (PC) and a constant in the instruction. RISC-V uses PC-relative addressing for both

conditional branches and unconditional jumps, because the destination of these

instructions is likely to be close to the branch.

* Why did RISC-V architects choose to encode the distance from the branch instruction to the branch target in *halfwords* (rather than bytes or words)?

//FIXME

* What are the four *addressing modes* in the RISC-V ISA?

1. Immediate addressing, where the operand is a constant within the instruction itself.
2. Register addressing, where the operand is a register.
3. Base or displacement addressing, where the operand is at the memory location whose address is the sum of a register and a constant in the instruction.
4. PC-relative addressing, where the branch address is the sum of the PC and a constant in the instruction.

* What are the key steps in translating and running a C program, and what system tools are used in the process?

C -> Compiled to assembly -> Assembled as machine code

* + In contrast, what are the steps when translating and running a Java program?

Java -> Interpreted to bytecodes -> Converted as machine code

* What is a *pseudoinstruction*, and what examples are given in our text?

A common variation of assembly language instructions often treated as if it were an instruction in its own right. Array in C is used to store elements of the same types whereas Pointers are address variables which store the address of a variable.

* Why can C code that uses pointers sometimes be more efficient than a version that uses arrays and array indices?

Sometimes an array uses more memory in the stack which is unnecessary.

* What additional RISC-V instructions are presented in Section 2.18?
  + What does the slt instruction do, and what other set instructions exist?

slt and sltu compare two registers as signed and unsigned numbers, respectively, then write 1 to a register if the first value is less than the second value, or 0 otherwise.

* + What does the auipc instruction do, and how might it be used?

Auipc is used for PC-relative memory addressing. Like the lui instruction, it holds a 20-bit constant that corresponds to bits 12 through 31 of an integer. auipc’s effect is to add this number to the PC and write the sum to a register. Combined with an instruction like addi, it is possible to address any byte of memory within 4 GiB of the PC.

* Why does having more powerful instructions in the instruction set not necessarily lead to better performance?

When it comes down to it, the complex pseudo code instructions that do more complex things are just a combination of other simpler lines of instructions. Creating more powerful instructions won't lead to better performance because complex instructions are just a combination of simpler ones.

* How likely is it that a human programmer can create more efficient code in assembly language than the compiler can generate from C code?

In theory yes, and sometimes humans can do it more efficiently by taking advantage of system facilities unknown to the compiler.

However, in practice it’s often a losing battle. Modern CPUs can be quite complex, and the actual efficiency of a sequence of instructions is dependent on the exact CPU things will run on.

**Chapter 4 - part A**

* Which factors in the "Classic CPU Performance Equation" are determined by the implementation of the processor?

Cycles Per Instruction (CPI), maybe the Cycles/Second (Clock Rate)

* What subset of RISC-V instructions are supported in the implementation presented by our text?

Memory reference Instructions: sw, lw

arithmetic-logical instructions add, sub, and, and or

conditional branch instruction branch if equal (beq)

* + What important instructions are not included in their design?

Shift, multiply, divide, and floating point instructions

* + What are the consequences in the design of implementing just a small subset of instructions?

Simpler instructions can reduce the Cycles per Instruction (CPI) of our computer. Complex math is very difficult for our processor

* + For each instruction included in our text's implementation, can you describe the sequence of steps required in its execution?

Sw:

Lw:

Add:

Sub:

And:

Or:

beq:

* + What processing steps do most RISC-V instructions have in common?

Instruction Fetch, Instruction Decode, Execute, Memory, Writeback

* What role does the *control unit* play in the execution of instructions?

It decides what signals go on and off (Memwrite, ALUSrc, loadPC) to execute the instruction properly

* What role does each of the 3 multiplexors play in the datapath shown in Figure 4.2?

ALUSrc: Controls whether Op2 comes from the immediate value or from a register

ALUOp/ALUCtrl: Controls what ALU operation is performed

PCSrc: Controls if a branch is

* + In Fig. 4.2, can you trace the execution of individual instructions?

Maybe

* + How many of the 5 classic components of a computer are shown in Fig. 4.2?

3: Datapath, Control, Memory (The other 2 are input and output, not depicted)

* + In Fig. 4.2, which parts of the datapath are *combinational elements*, and which are *state elements*?

Comb: ALUSrc, ALUCtrl, Memread/Write?

State: Register Read/Write, PC update MUX

* + If execution were halted at some point in time, what information would be required to resume execution later at that point?

The PC counter? The return address, in X1

* What inputs does every *state* element have?

1- Data value to be written to the element, and 2- the clock

* + What role does the clock play in the operation of the datapath?

It changes the value of the PC, which moves the program along. It also allows values to be updated in the register, I think.

* + What is a *clocking methodology*?

Defines when signals can be read and when they can be written

* + If a design uses positive edge-triggered state elements, what happens only on rising clock edges?

State Changes

* In the datapath, where must all inputs to combinational logic come from?

From a set of state elements (synchronous)

* + In the datapath, where must all outputs to combination logic go to?

To a set of state elements (synchronous)

* In Figure 4.3, what determines the length of a clock cycle?

The worst-case time to get through combinational logic between two elements in the processor

* In general, when does a state element require an explicit write control signal?

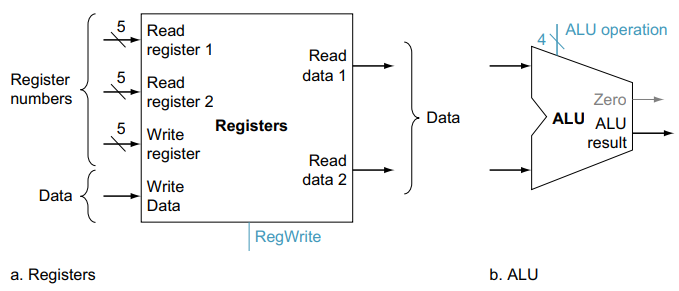
If a state element is not updated on every clock, then an explicit write control signal is required

* For the design in Fig. 4.4, what prevents feedback within a single clock cycle?

Feedback cannot occur within one clock cycle because of the edge-triggered

update of the state element.

* In figures throughout Chapter 4, what is the significance of signals drawn and labeled in blue?



The blue signals are usually ones that are like a mux where they determine what the block does. They are “Ctrl signals”

* + How do they differ from signals drawn and labeled in black?

Signals in black are the general input and output signals and are usually multi-bit.

* Since a register file may be read and written in the same cycle, does a datapath that uses edge-triggered writes require multiple copies of the register file?

No

* At the start of any clock cycle, what value is in the PC?

Fix Me

* + Why isn't the ALU used to increment the PC?

The PC is not one of the inputs to the ALU, and therefore could not be added to

* What input(s) must be provided to the register file to read or write a given register?

Read Register 1, Read Register 2, Write Register, Write Data, RegWrite

* + What are the consequences of the register file having a *write control* signal but no *read control* signal?

Registers are automatically read to the ALU (Register 1 is always read, Register 2 is read if ALUSrc is off). They are also read when MemWrite is on.

* + For the RISC-V datapath, how many ports must the register file have? Why?

8 Ports. Clk and write: 1 bit each. readReg1, readReg2, writeReg: 5 bits each. WriteData, readData1, readData2: 32 bits each. Source - the reg file we wrote for lab 3.

* + How is it possible for the register file to read and write the same register in the same clock cycle?

If either read register = the write register, we just send the write value to the read register on that clock cycle

* + If a read and write occur to the same register in the same clock cycle, what value does the read return?

The read data gets the current write data

* Why does the data memory require both read and write signals, unlike the register file?

The Register file depends on other signals to be read. The memory’s read and write signals are meant to stop it from being read and written to except when it is specifically necessary.

* What is the 32-bit input to the Imm Gen unit?

The entire instruction

* + What does the Imm Gen unit do?

Extracts and Sign-extends immediate

* + How does the Imm Gen unit know how the immediate value is encoded?

Uses the opcode

* Why isn't the ALU used to compute the *branch target address* for branch instructions?

This calculation needs the PC. The ALU doesn’t get the PC

* + When is the "Zero" output of the ALU asserted?

When the result of the ALU operation = zero.

* + Why does the branch offset have to be shifted left by one bit, and what hardware is required to do it?

The Branch offset in the instruction does not include bit 0, which is always 0. So we shift left by one to add a 0 on the right end.

* Why are some elements in the RISC-V datapath replicated?

This allows for parallelism in a processor, which allows multiple instructions to be issued at the same time.

* In the RISC-V datapath, what are the two possible values that could be used to update the PC at the end of each cycle?

PC = PC + 4

PC = PC + offset

* + What are the two possible values that could be used to update a register in the register file at the end of each cycle?

Read data (from memory) or

ALU Result

* + What are the two possible values that could appear on the lower input to the ALU in each cycle?

Read 2 (from Register file) or

the immediate value

* + For any given RISC-V instruction, can you determine which input each MUX should pass through to execute that instruction correctly?

Yeah Probably

* Which RISC-V instructions are covered by the "simple" datapath and control implementation described in Sections 4.3 and 4.4 of the text?

And, Or, Add, Subtract

* What do each of the 7 control signals do that are generated by the Control unit? (See Fig. 4.17)

RegWrite: Write to a register

ALUSrc: Op 2 of ALU is Read 2 or Immediate

PCSrc: Branch offset or + 4

ALUCtrl/ALUOp: Decides what ALU Operation to use

MemWrite: Write a value from memory

MemRead: Update Memory read value

MemtoReg: Only for Load instructions. Load a value from memory to Register

* + What control signals does the datapath require that cannot be generated using only the 7 opcode bits?

ALUCtrl and PCSrc

* What control signal in Fig. 4.17 is generated using multiple levels of decoding?
  + What is the main motivation for using multiple levels of control logic?
  + How many bits wide is the ALUOp value generated by the first level of control logic?

2

* + What additional inputs are used in the second level of control logic?

Funct 3, Funct 7

* + How many different functions can the ALU perform in the simple implementation in the text?

4?

* + How many bits wide is the input to the ALU in the simple implementation in the text?

4

* + For each of those instructions, can you specify the value of each of the control signals?

|  | RegWrite | MemRead | MemWrite | ALUSrc | MemtoReg | PCSrc |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction:  (and) | 1 | 0 | 0 | 0 | 0 | 0 |
| Load(ld), Arith, Imm: | 1 | 1 | 0 | 1 | 1 | 0 |
| Store(sd): | 0 | 0 | 1 | 1 | X | 0 |
| Cond., Branch(beq): | 0 | 0 | 0 | 0 | X | 1 |
| Uncond., Jump: | 0 | 0 | 0 | 0 | X |  |
| Load Upper Imm.(lui): | 1 | 0 | 0 | 0 |  | 0 |
| Unique Inputs: |  |  |  |  |  | 0 |

* + For each of those instructions, how long does it take to execute?

//FIXME

* + In this design, can some instructions execute more quickly than others?

Yes

* In what way does the single-cycle implementation violate the principle of making the common case fast?

The penalty for using the single-cycle design with a fixed clock cycle is significant, but might be considered acceptable for this small instruction set. Historically, early computers with very simple instruction sets did use this implementation technique. However, if we tried to implement the floating-point unit or an instruction set with more complex instructions, this single-cycle design wouldn’t work well at all. Because we must assume that the clock cycle is equal to the worst-case delay for all instructions, it’s useless to try implementation techniques that reduce the delay of the common case but do not improve the worst-case cycle time.

