目录：

Chapter 2:

1. ISA
   1. LC-2200 and LC-900 from project 1
   2. Addressing Modes
   3. Instruction Types
   4. CISC/RISC
   5. Endianness, Alignment, Reordering
2. Calling Convention

Chapter 3:

1. Combinational vs. Sequential Logic
2. Datapath
   1. Construction
   2. Microcode
   3. Macro States
3. Controller
4. Instruction set design from High Level Language
   1. Expressions, assignments => ALU instructions
   2. Data abstraction => Addressing modes
   3. Conditional & loop statements => Branch
   4. Procedure calls/returns => stack management
5. Where operands stored?
   1. Memory
      1. When the size of memory grows, the size of memory also goes up. This may cause the addressability issue.
      2. Memory is expensive
   2. Register
6. Immediate values: constant values be part of the instruction itself. Handle in compiling HHL.
7. Operand Granularity:
   1. 1 char = 8 bits = 1 byte
   2. 1 short = 16 bits = 1 half word
   3. 1 int = 32 bits = 1 word (LC-2200 addressability)
   4. 1 long = 64 bits
8. Most Significant Byte (MSB) & Least Significant Byte (LSB)
   1. MSB 在最左面 LSB 在最右面
9. Endianness: Ordering of the bytes within the word.
   1. 0x11223344
   2. Big endian:
      1. The MSB of the word is at the address of the word operand
      2. 0x11
   3. Little endian:
      1. The LSB of the word is at the address of the word operand
      2. 0x44
   4. Network codes use format conversion routines between host to network format, and vice versa, to avoid correctness issues caused by endianness.
10. Alignment:
    1. **Pack** operands: laying out the operands in memory ensuring no wasted space (not always the right approach)
       1. 第一目标节省空间，如果能将几个element合并成行就合并
       2. 尽量保证统一element不夸行，如遇到合并会导致夸行的话就分行
          1. It will cause inefficient if we pack in this situation.
11. Address mode: refers to the way the operands are specified in an instruction
    1. Register addressing mode:
       1. Operands stored in register.
       2. Data abstraction
    2. Base + offset mode: ld r\_t, offset(r\_base)
       1. Memory address us computed in the instruction as the sum of the contents of a register in the processor (base register) and offset (immediate value) from that register.
       2. It can be used to load / store simple variables and also elements of compound variables.
    3. Base + index mode:
       1. Addressing mode to let the effective address be computed as the sume of the contents of two registers.
       2. Use for array
    4. PC-addressing mode
    5. Indirect addressing mode: for conditionals and loops
       1. ld @ra
    6. Pseudo-direct addressing mode
       1. Address is formed from first 6 bits of PC and last 26 bits of instruction.
12. Switch vs. if
    1. Switch can use jump table: no need to evaluate one by one.
13. Function call
    1. State of caller
    2. Pass parameters to callee
    3. Remember return address (JALR r\_targer , r\_link; r\_link <= PC, PC <= r\_target)
    4. Jump to procedure
    5. Allocate space for local vars
    6. Pass result to caller and return to caller
    7. Save the result and continue the program
14. Save state
    1. Stack
    2. Shadow register sets: no memory accesses (fast, but need lots of extra registers)
15. Register in LC-2200
    1. s0 to s2: caller’s source register (callee must preserve if it wants to use them)
    2. t0 to t2: temporary register (caller must preserve if it wants their values to persist over a function call)
    3. a0 to a2: parameter passing registers
    4. v0: return value
    5. ra: return address
    6. at: target address
    7. sp: stack pointer
    8. fp: frame pointer
16. Calling Convention
    1. Step 1: caller saves any of registers t0-t2 on the stack(if it needs the values in them upon return)
    2. Step 2: caller places the parameters in a0-a2(using the stack for additional parameters if needed)
    3. Step 3: caller allocates space for any additional return values on the stack
    4. Step 4: caller saves previous return address currently in ra
    5. Step 5: caller executes JALR at, ra
    6. Step 6: callee saves frame pointer on the stack and sets frame pointer to the stack pointera
    7. Step 7: callee saves any of registers s0-s2 that it plans to use during its execution on the stack.
    8. Step 8: callee allocates space for any local variables on the stack
    9. Step 9: prior to return, callee restores any saved s0-s2 registers from the stack
    10. Step 10: Callee replaces the value in sp with the value from fp to pop the local variables and saved s registers off the stack.
    11. Step 11: callee executes jump to ra
    12. Step 12: upon return, caller restores previous return address to ra
    13. Step 13: caller stores additional return values as desired
    14. Step 14: upon return, caller moves stack pointer to discard additional parameters
    15. Step 15: upon return caller restores any saved t0-t2 registers from the stack.
17. Instruction Formats:
    1. Zero operand
       1. HALT, NOP
       2. Stack machines: add, sub, push, pull
    2. One operand
       1. INC/DEC/NEG/NOT/J
       2. Load M, Add M
    3. Two operands
       1. add r1, r2
       2. mov r1, r2
    4. three operands
       1. add r1, r2, r3
       2. load rd, rb, offset
    5. fixed length instructions
       1. pro
          1. simplifies implementation
          2. can start interpreting
       2. con
          1. may waste space
          2. may need additional logic in data path
          3. limits instruction set
    6. variable length instructions
       1. pro
          1. no wasted space
          2. less constraints on designer
          3. more flexibility with opcodes, addressing modes and operands
       2. con
          1. complicates implementation
18. CISC & RISC
    1. CISC (Complex instruction set computing): has the ability to execute addressing modes or multi-step operations within one instruction set
       1. Emphasis on hardware
       2. Includes multi-clock complex instructions
       3. Memory to memory, ld and st incorporated in instructions
       4. Small code sizes, high cycles per second
       5. Transistors used for storing complex instructions
    2. RISC (ARM, LC-2200)
       1. Emphasis on software
       2. Single-clock, reduce instruction only
       3. Register to register, ld and st are independent instructions
       4. Low cycles second large code sizes
       5. Spends more transistors on memory registers
19. Instruction type: (unused should be 0)
    1. R-type (add, nand)
       1. A screenshot of a social media post

          Description automatically generated
       2. Register addressing mode.
    2. I-type (addi, lw, sw, beq)
       1. A screenshot of a cell phone

          Description automatically generated
       2. PC-Relative addressing mode.
       3. Base + Offset addressing mode.
       4. Immediate.
    3. J-type (jalr)
       1. A screenshot of a cell phone

          Description automatically generated
       2. Register addressing mode.
    4. O-type (halt)
       1. A picture containing drawing

          Description automatically generated
       2. None addressing mode.
20. LC-2200 instruction sets
    1. A screenshot of a cell phone

       Description automatically generated
21. LC-2200 registers
    1. A screenshot of a cell phone

       Description automatically generated
    2. s register saved only if needed
    3. fp register always save
22. Issues influencing processor design
    1. OS
    2. Programming language support
    3. Memory system
    4. Parallelism
    5. Debugging
    6. Virtualization
    7. Fault tolerance
    8. Security
23. Logic triggering
    1. Level triggering
       1. Outputs change based on inputs whenever clock is high
       2. Memory will be considered to be level triggered (cost reason)
    2. Edge triggering
       1. Outputs change based on inputs only when clock transitions
       2. Positive edge-triggered logic when leading edge cause triggering
       3. Negative edge-triggered when trailing edge causes triggering
       4. Register is edge-triggered!!!
24. Combinational vs Sequential logic:
    1. Combinational (ALU, Garage Door Controller)
       1. For a given set of inputs there is one unique output
    2. Sequential
       1. Circuits contain elements that remember state
       2. Outputs depends on combinational logic that consider circuit inputs and previous state
25. Delays:
    1. Clock width >= max (the sum of all data path delays)
26. Bus-based design
    1. Must make connections between circuit elements for every instruction
    2. Numerous connections are expensive and take up valuable space
    3. Have a set of vires that all elements can connect to and share in order to transfer information
27. Hardware truth table: ROM (store all the truth table in FSM)
    1. Address in ROM: input bits
    2. Contents of ROM: output bits
28. Alternative style of control unit design
    1. Flat ROM
       1. More space
       2. Faster since only one ROM access in each microinstruction
    2. Micro sequencer (3-ROM control unit)
       1. Less space (main ROM much smaller than Flat ROM
       2. Slower since additional ROM access in every clock cycle