Discussion 3

Jan 22, 2024

Outline

1. DINOCPU assignment2
    a. *auipc* instruction
    b. Memory instruction
    c. Branch instruction
2. Week 3 quiz

```
/**
 * Main control logic for our simple processor
 *
 * Input: opcode:               Opcode from instruction
 *
 * Output: aluop               Specifying the type of instruction using ALU
 *                                  . 0 for none of the below
 *                                  . 1 for arithmetic instruction types (R-type or I-type)
 *                                  . 2 for non-arithmetic instruction types that uses ALU (auipc/jal/jarl/Load/Store)
 * Output: arth_type           The type of instruction (0 for R-type, 1 for I-type)
 * Output: int_length          The integer length (0 for 64-bit, 1 for 32-bit)
 * Output: jumpop              Specifying the type of jump instruction (J-type/B-type)
 *                                  . 0 for none of the below
 *                                  . 1 for jal
 *                                  . 2 for jalr
 *                                  . 3 for branch instructions (B-type)
 * Output: memop                Specifying the type of memory instruction (Load/Store)
 *                                  . 0 for none of the below
 *                                  . 1 for Load
 *                                  . 2 for Store
 * Output: op1_src              Specifying the source of operand1 of ALU/JumpDetectionUnit
 *                                  . 0 if source is register file's readdata1
 *                                  . 1 if source is pc
 * Output: op2_src              Specifying the source of operand2 of ALU/JumpDetectionUnit
 *                                  . 0 if source is register file's readdata2
 *                                  . 1 if source is immediate
 *                                  . 2 if source is a hardwired value 4
 * Output: writeback_src        Specifying the source of value written back to the register file
 *                                  . 0 if writeback is invalid
 *                                  . 1 to select alu result
 *                                  . 2 to select immediate generator result
 *                                  . 3 to select data memory result
 * Output: validinst            0 if the instruction is invalid, 1 otherwise
 *
 * For more information, see section 4.4 of Patterson and Hennessy.
 * This follows figure 4.22.
 */
```

beq rs1, rs2, offset

(imm)

auipc Instruction

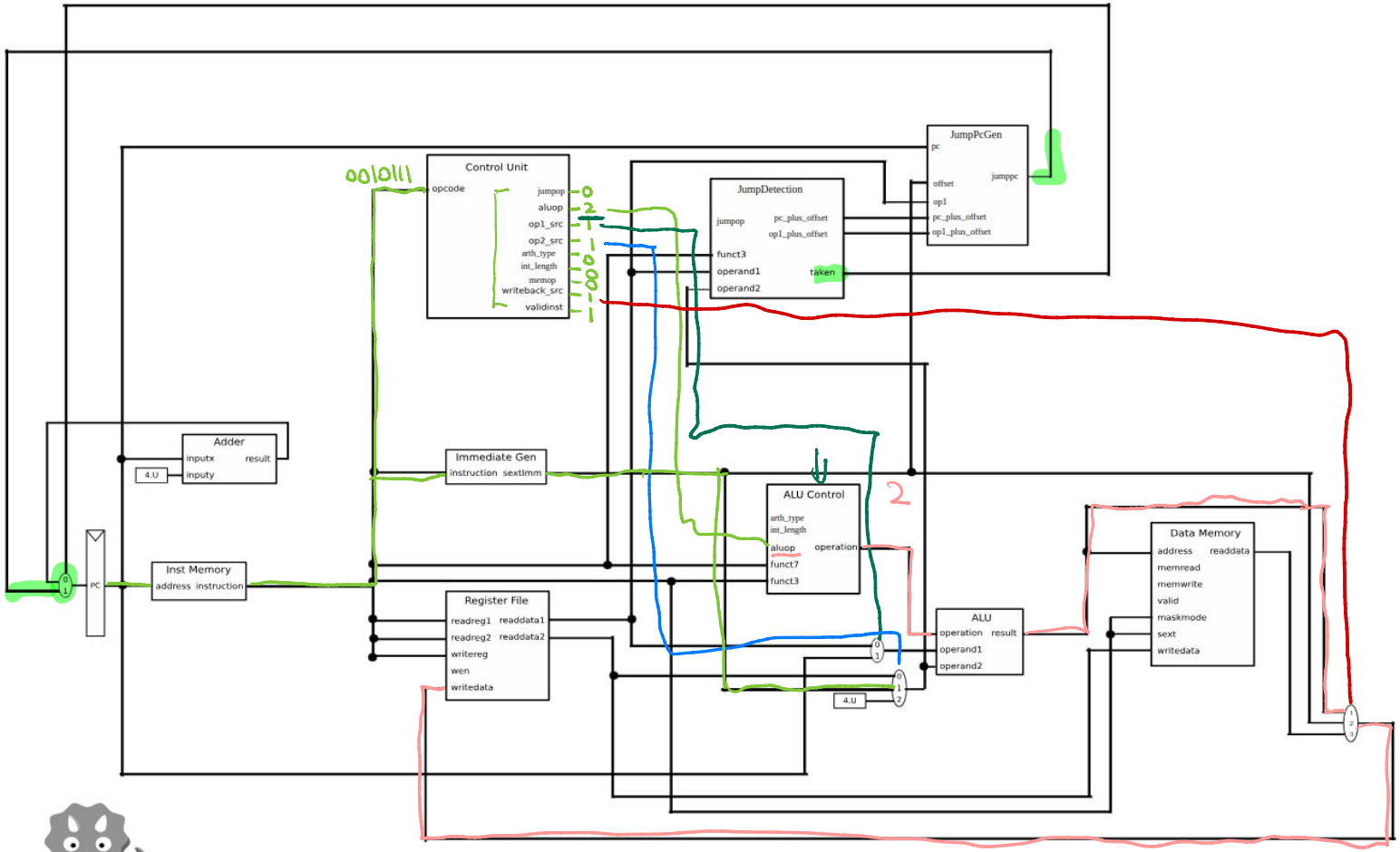
`auipc` instruction details

The following table shows how the `auipc` instruction is laid out.

31-12	11-7	6-0	Name
imm[31:12]	rd	0010111	auipc

`auipc` stands for "add upper immediate to pc". The instruction has the following effect,

```
R[rd] = pc + imm << 12
```



Pipelined DINO CPU

Memory Instruction

31-25	24-20	19-15	14-12	11-7	6-0	Name
imm[11:5]	imm[4:0]	rs1	000	rd	0000011	lb
imm[11:5]	imm[4:0]	rs1	001	rd	0000011	lh
imm[11:5]	imm[4:0]	rs1	010	rd	0000011	lw
imm[11:5]	imm[4:0]	rs1	011	rd	0000011	ld
imm[11:5]	imm[4:0]	rs1	100	rd	0000011	lbu
imm[11:5]	imm[4:0]	rs1	101	rd	0000011	lhu
imm[11:5]	imm[4:0]	rs1	110	rd	0000011	lwu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	sw
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	sd

byte , halfword , word , doubleword
8 , 16 , 32 , 64.

signed.

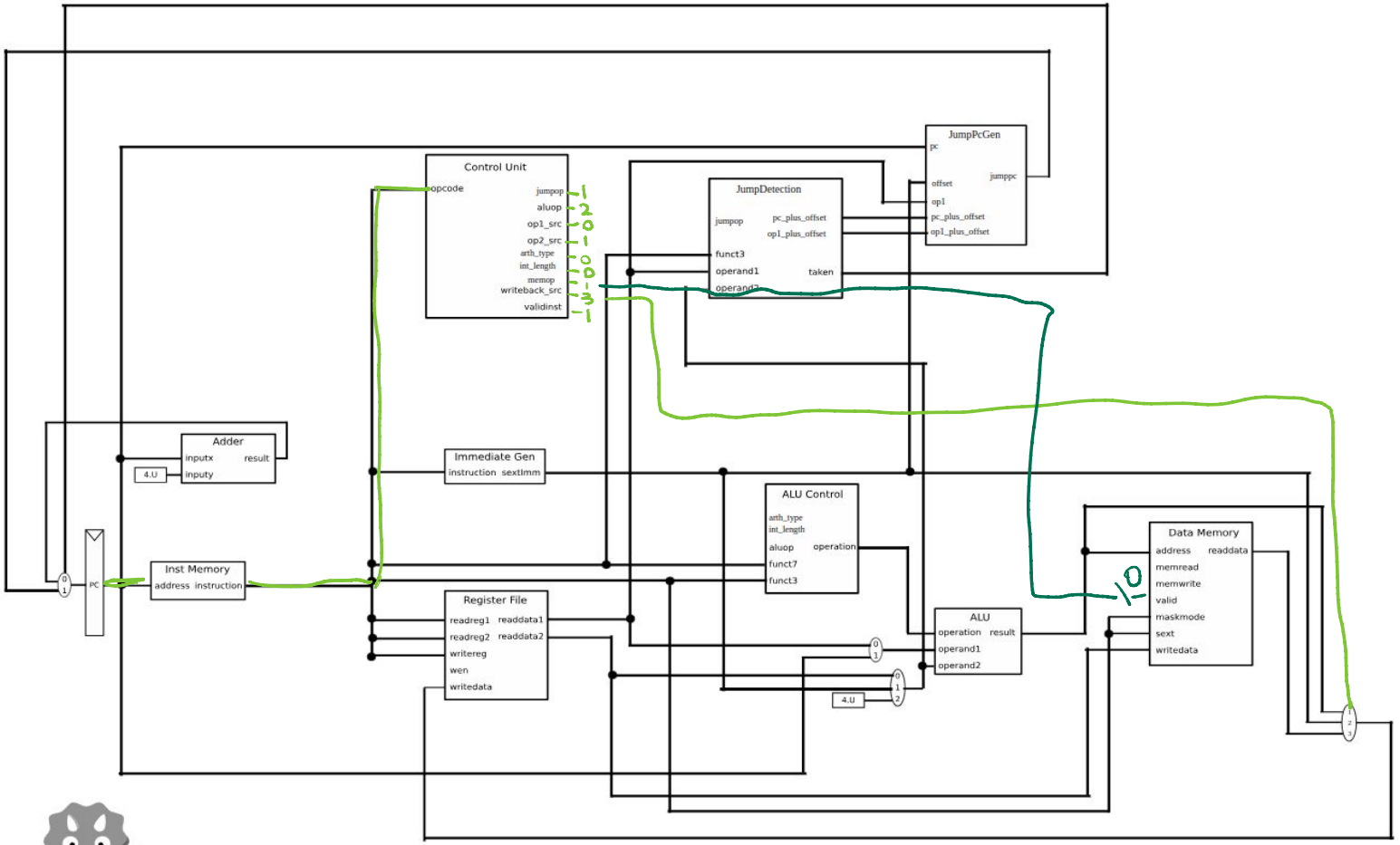
Load

unsigned.

store.

```scala
/**
 * The *interface* of the DMemPort module.
 *
 * Pipeline <=> Port:
 *   Input:  address, the address of a piece of data in memory.
 *   Input:  writedata, valid interface for the data to write to the address
 *   Input:  valid, true when the address (and writedata during a write) specified is valid
 *   Input:  memread, true if we are reading from memory
 *   Input:  memwrite, true if we are writing to memory
 *   Input:  maskmode, mode to mask the result. 0 means byte, 1 means halfword, 2 means word, 3 means doubleword
 *   Input:  sext, true if we should sign extend the result
 *   Output: readdata, the data read and sign extended
 *   Output: good, true when memory is responding with a piece of data
 */
class DMemPortIO extends MemPortIO {
  // Pipeline <=> Port
  val writedata = Input(UInt(64.W))
  val memread   = Input(Bool())
  val memwrite  = Input(Bool())
  val maskmode  = Input(UInt(2.W))
  val sext      = Input(Bool())

  val readdata  = Output(UInt(64.W))
}
```



Pipelined DINO CPU

Branch Instruction

imm[12, 10:5]	rs2	rs1	funct3	imm[4:1, 11]	opcode	Name
31-25	24-20	19-15	14-12	11-7	6-0	
imm[12, 10:5]	rs2	rs1	000	imm[4:1, 11]	1100011	beq
imm[12, 10:5]	rs2	rs1	001	imm[4:1, 11]	1100011	bne
imm[12, 10:5]	rs2	rs1	100	imm[4:1, 11]	1100011	blt
imm[12, 10:5]	rs2	rs1	101	imm[4:1, 11]	1100011	bge
imm[12, 10:5]	rs2	rs1	110	imm[4:1, 11]	1100011	bltu
imm[12, 10:5]	rs2	rs1	111	imm[4:1, 11]	1100011	bgeu

op.asUInt.
op.asSInt.

rs1 == rs2
rs1 =\= rs2
rs1 < rs2
rs1 ≥ rs2.

unsigned.

```
/**
 * JumpDetection Unit.
 * This component takes care of deciding the PC of the next cycle upon a jump instruction (jump/branch-type).
 *
 * Input: jumpop        Specifying the type of jump instruction (J-type/B-type)
 *                                    . 0 for none of the below
 *                                    . 1 for jal
 *                                    . 2 for jalr
 *                                    . 3 for branch instructions (B-type)
 * Input: operand1                    First input
 * Input: operand2                    Second input
 * Input: funct3                      The funct3 from the instruction
 *
 * Output: pc_plus_offset             True if the next pc is the current pc plus the offset (imm)
 * Output: op1_plus_offset            True if the first operand is the first operand plus the offset (imm)
 * Output: taken                      True if, either the instruction is a branch instruction and it is taken, or it is a jump instruction
 *
 */
class JumpDetectionUnit extends Module {
  val io = IO(new Bundle {
    val jumpop            = Input(UInt(2.W))
    val operand1          = Input(UInt(64.W))
    val operand2          = Input(UInt(64.W))
    val funct3            = Input(UInt(3.W))

    val pc_plus_offset    = Output(Bool())
    val op1_plus_offset   = Output(Bool())
    val taken             = Output(Bool())
  })
```

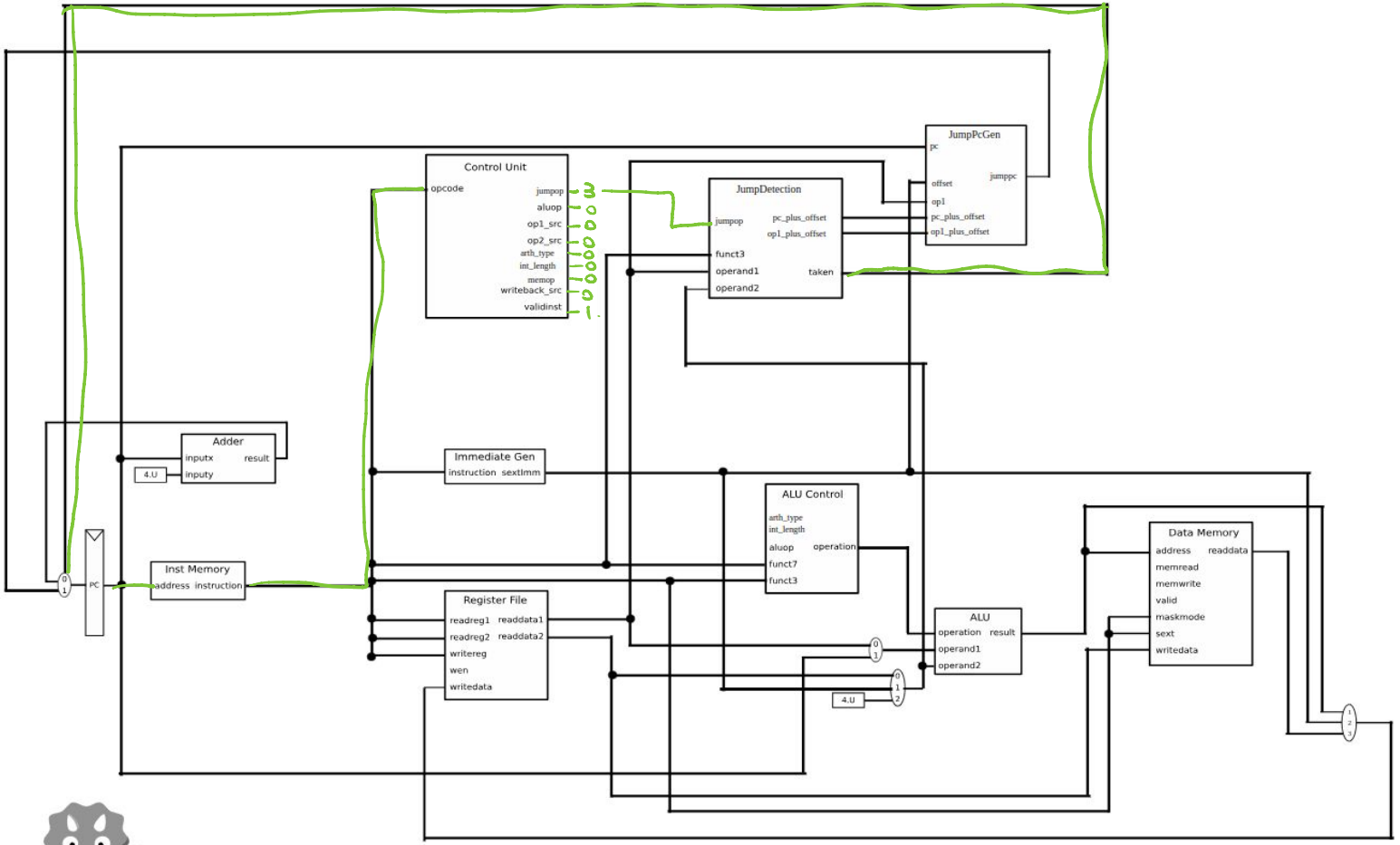
```
/**
 * JumpPcGenerator Unit.
 * This component takes care of calculating the pc that the jump instruction is jumping to.
 *
 * Input: pc_plus_offset          True if the next pc is the current pc plus the offset (imm)
 * Input: op1_plus_offset         True if the first operand is the first operand plus the offset (imm)
 * Input: pc                      The PC of the current instruction
 * Input: op1                     The first operand of the current instruction
 * Input: offset                  The offset (imm) of the current instruction
 *
  * Output: jumppc               The pc that the jump instruction is jumping to
 *
 */
class JumpPcGeneratorUnit extends Module {
  val io = IO(new Bundle {
    val pc_plus_offset    = Input(Bool())
    val op1_plus_offset   = Input(Bool())
    val pc                = Input(UInt(64.W))
    val op1               = Input(UInt(64.W))
    val offset            = Input(UInt(64.W))

    val jumppc            = Output(UInt(64.W))

  })

  // default case, i.e., not a jump instruction
  io.jumppc := 0.U

  when (io.pc_plus_offset) {
    io.jumppc := io.pc + io.offset
  }
  .elsewhen (io.op1_plus_offset) {
    io.jumppc := io.op1 + io.offset
  }
}
```



Pipelined DINO CPU

Reminders

1. Use “instruction”, don’t use “imem.io.instruction”
2. Don’t modify the source code of “JumpPcGeneratorUnit”
3. The immediate generator will produce the shifted and sign extended value! You do not need to shift the immediate value outside of the immediate generator.

4). .asSInt for signed int.
.asUInt for unsigned int.

Week 3 Quiz

Like the last quiz, we're going to be comparing the same two systems: The AMD Epyc and Intel i7.

I've run a few other SPEC workloads on these two systems.

	AMD Epyc	Intel i7
gcc	274.3s	180.0s
mcf	301.1s	186.3s
libquantum	313.1s	230.4s

Use this info for the questions below.

Question 1

2 pts

For mcf, what is the Speedup of the Intel i7 compared to the AMD Epyc?

Speedup = $\frac{\text{old time}}{\text{new time}} = \frac{301.1}{186.3}$

$= 1.61x$.

Like the last quiz, we're going to be comparing the same two systems: The AMD Epyc and Intel i7.

I've run a few other SPEC workloads on these two systems.

	AMD Epyc	Intel i7
gcc	274.3s	180.0s
mcf	301.1s	186.3s
libquantum	313.1s	230.4s

= 1.523
= 1.616
= 1.358

Use this info for the questions below.

Question 2

2 pts

For which application does the i7 get the greatest speedup?

- ✓ mcf
- libquantum
- gcc

Like the last quiz, we're going to be comparing the same two systems: The AMD Epyc and Intel i7.

I've run a few other SPEC workloads on these two systems.

	AMD Epyc	Intel i7
gcc	274.3s	180.0s
mcf	301.1s	186.3s
libquantum	313.1s	230.4s

Use this info for the questions below.

$\sqrt[3]{274.3 \times 301.1 \times 313.1}$

$\sqrt[3]{180 \times 186.3 \times 230.4}$

$= 1.495$

Question 3

2 pts

What is the average speedup for these three applications?

Hint: Use the *correct* average statistic. See section 1.9 in the book.

Question 4

2 pts

You are a computer architect working at a startup. Your marketing department says "If we want to succeed, we need to get a 1.8x speedup compared to our competition."

Unfortunately, you and your competitor use the same foundry and will end up with about the same frequency and you're using the same ISA as your competitor.

So, you only have control over the microarchitecture. How much "improvement" in IPC (instructions per cycle which is 1/CPI) is required for you to meet marketing's goal?

Iron law exec. time = #inst x CPI x cycletm

- ○ 1.2x
- ○ 2.2x
- ✓ 1.8x
- ○ 1.4x

$\Rightarrow \frac{1}{1.8x}$

$IPC = \frac{1}{CPI} \Rightarrow 1.8x$

Question 5

2 pts

You are working on a new generation processor with a new ISA, a new microarchitecture, and a new manufacturing process. This new architecture will allow you to increase the frequency by 1.5x, but it requires increasing the number of instructions by 1.0x. Since these are simpler instructions, you've found a way to decrease the CPI from 3 to 1.2. What is the overall speedup of this new design?

$$\text{Speedup} = \frac{\text{old}}{\text{new}} = \frac{\#\text{inst} \times 3 \times \frac{1}{f}}{\#\text{inst} \times 1.2 \times \frac{1}{1.5 \times f}}$$

$$= \frac{3}{1.2 \times \frac{1}{1.5}} = 3.75$$

Question 6

2 pts

The ISA is the contract between the [Select] and the [Select].

[Select]
application
runtime
microarchitecture
hardware

[Select]
operating system
software
engineers
programming language

Question 7

2 pts

Which of the following is part of the ISA?

- [x] Virtual memory
- [x] Size of registers
- [ ] How to implement the hardware (e.g., pipelined, number of ALUs, etc.
- [x] Number of registers
- [x] Instruction format

Question 8 — 2 pts

RISC ISAs usually have [Select] different instructions defined compared to CISC ISAs

- [Select]
- fewer
- more

Question 9 — 2 pts

A program compiled for a RISC ISA will probably dynamically execute [Select] instructions than if it was compiled for a CISC ISA.

- [Select]
- fewer
- more

Question 10 — 2 pts

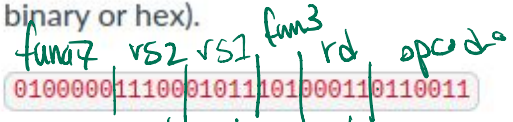
One of the main in technology which drove the industry to move from CISC ISAs to RISC ISAs is that high-level language compiler technology improved.

- True
- False

Question 11

2 pts

Decode the registers for the following R-type instruction. Give your answers in decimal (not binary or hex).



MSB <--> LSB

Source register 1:

Source register 2:

Destintation register:

Question 12

2 pts

31 − 12 [20, 10:1,

The following instruction is a JAL instruction. What is the sign of the immediate value?

`00000110010111011101011011101111`

MSB <--> LSB

- ○ negative
- ✓ positive

Question 13

1 pts

Which of the following characteristics of the RISC-V ISA makes it simpler to implement in hardware than a CISC ISA?

- [ ] There are extensions so customers can add their own instructions.
- [ ] It has many different kinds of R-type instructions.
- [ ] The destination register is always in the same location in the instruction.
- [ ] The instructions are all the same width (32 bits).

Question 14

1 pts

The JAL instruction is used for...

- ◯ Memory operations
- ◯ Conditional statements
- ◯ Function calls
- ◯ System calls
- ◯ Simple arithmetic operations
- ◯ Loading immediate values into the register file

Question 15

2 pts

" This time, let's encode an instruction instead of decoding.

Given the following assembly, choose the correct binary representation.

```
sub x30 x3 x9
```

MSB <--> LSB

- `01000000100100011011111100110011`
- `01101101100100011000111100000011`
- `01000001010111011011111100110011`
- `01000000100100011000111100110011`