CSSE 232 Ccover page including your team name and team member's names.OMP ARCH 1 DESIGN DOCUMENT

TEAM:

Drew Kilner, Yash Anand, Julian Ferrer Rodriguez, Eli Granade, Ziyu Xie

Euclid's algorithm in RISC-V assembly:

```
relPrime:   addi sp, sp, -16
            sw ra, 0(sp)
            sw s0, 4(sp)
            sw s1, 8(sp)
            addi s0, x0, 2
            addi s1, x0, 1
            sw a0, 12(sp)
loop:       addi a1, s0, 0
            lw a0, 12(sp)
            jal ra, gcd
            beq a0, s1, done
            addi s0, s0, 1
            jal x0, loop
done:       addi a0, s0, 0
            lw ra, 0(sp)
            lw s0, 4(sp)
            lw s1, 8(sp)
            addi sp, sp, 16
            jalr x0, ra
```

```
gcd:        addi sp, sp, -4
            sw ra, 0(sp)
            beq a0, x0, returnb
loop:       beq a1, x0, returna
            blt a1, a0, agreater
            sub a1, a1, a0
            jal x0, loop
agreater:   sub a0, a0, a1
            jal x0, loop
returnb:    addi a0, a1, 0
returna:    lw ra, 0(sp)
            addi sp, sp, 4
            jalr x0, ra
```

Required Instructions:

```
addi rd, rs, imm
sub rd, rs1, rs2
sw rd, imm (rs)
lw rd, imm (rs)
beq rs1, rs2, imm
blt rs1, rs2, imm
jal rd, imm
jalr rd, imm
```

Optimization:

Do not assign zero register but rather use value 1 in `x0`. Enables imm values for stack frame building by using shift instructions. Also enables beq without first assigning value to register. Introducing, bez (branch equal zero) instruction.

Instructions:

inst	fmt	func	opcode	description
add	R	00	000	R[rd] = R[rs1] + R[rs2]
sub	R	01	000	R[rd] = R[rs1] – R[rs2]
or	R	10	000	R[rd] = R[rs1] | R[rs2]
and	R	11	000	R[rd] = R[rs1] & R[rs2]
addi	I	00	001	R[rd] = R[rs1] + SE(imm)
sli	I	01	001	R[rd] = R[rs1] << imm[4;
sri	I	10	001	
xori	I	11	001	
lw	M		010	
sw	M		011	
beq	B		100	
blt	B		101	
jal	J		110	
jalr	J		111	

15 14	13 12	11 9	8 6	5 3	2 0	Type
func		rs2	rs1	rd	op	R
func	imm[4:0]		rs1	rd	op	I
imm[6:3]		rs2	rs1	imm[2:0]	op	B
imm[6:0]			rs1	rd	op	M
imm[9:0]				rd	op	J

- A cover page including your team name and team member's names.

  Done

- A one paragraph description of your design at a high level including your design philosophy.

Load and store design mimicking RISC-V at first to ensure full functionality with the same design philosophies. After full functionality is achieved, optimize for performance (defined below) of the benchmark version of Euclid's Algorithm. New instructions specifically optimized for this task seems the most direct. This can later result in the elimination of other instructions (given processor still is general purpose), which satisfies the simplicity rule. Combination with accumulator is a stretch goal, if with proven contribution to performance, or with extra time.

- A description of how you plan on measuring "performance" of your processor.

Time it takes to run Euclid's Algorithm for the same numbers.

Copying the green sheet (format) completes the tasks between the lines

- Description of the registers available to the assembly language programmer and those reserved for any specific purpose. This is like the register list from the green sheet, notice that PC and other such registers are not included here.
- A symbolic description of the behavior of each instruction (like those on the green sheet).

Same as "description" on green sheet

- You should summarize the ISA in a table like the one on the green sheet. This table should include (at minimum) instruction mnemonic, type, opcode, and symbolic description.
- You should draw a memory map allocating space for text, data, and any special addresses your design needs.

- An unambiguous English description of each machine language instruction format type and its semantics including a visual depiction of the allocation of bits in each instruction type (like on the green sheet).

- An unambiguous English description of the syntax and semantics of each instruction (see Figure 2.1 on page 69 of the book).

RISC-V operands

Name	Example	Comments
32 registers	x0-x31	Fast locations for data. In RISC-V, data must be in registers to perform arithmetic. Register x0 always equals 0.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4,294,967,292]	Accessed only by data transfer instructions. RISC-V uses byte addresses, so sequential word accesses differ by 4. Memory holds data structures, arrays, and spilled registers.

RISC-V assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	`add x5, x6, x7`	`x5 = x6 + x7`	Three register operands; add
	Subtract	`sub x5, x6, x7`	`x5 = x6 - x7`	Three register operands; subtract
	Add immediate	`addi x5, x6, 20`	`x5 = x6 + 20`	Used to add constants
Data transfer	Load word	`lw x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Word from memory to register
	Load word, unsigned	`lwu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned word from memory to register
	Store word	`sw x5, 40(x6)`	`Memory[x6 + 40] = x5`	Word from register to memory
	Load halfword	`lh x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Halfword from memory to register
	Load halfword, unsigned	`lhu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned halfword from memory to register
	Store halfword	`sh x5, 40(x6)`	`Memory[x6 + 40] = x5`	Halfword from register to memory
	Load byte	`lb x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte from memory to register
	Load byte, unsigned	`lbu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte unsigned from memory to register
	Store byte	`sb x5, 40(x6)`	`Memory[x6 + 40] = x5`	Byte from register to memory
	Load reserved	`lr.d x5, (x6)`	`x5 = Memory[x6]`	Load; 1st half of atomic swap
	Store conditional	`sc.d x7, x5, (x6)`	`Memory[x6] = x5; x7 = 0/1`	Store; 2nd half of atomic swap
	Load upper immediate	`lui x5, 0x12345`	`x5 = 0x12345000`	Loads 20-bit constant shifted left 12 bits
Logical	And	`and x5, x6, x7`	`x5 = x6 & x7`	Three reg. operands; bit-by-bit AND
	Inclusive or	`or x5, x6, x8`	`x5 = x6 | x8`	Three reg. operands; bit-by-bit OR
	Exclusive or	`xor x5, x6, x9`	`x5 = x6 ^ x9`	Three reg. operands; bit-by-bit XOR
	And immediate	`andi x5, x6, 20`	`x5 = x6 & 20`	Bit-by-bit AND reg. with constant
	Inclusive or immediate	`ori x5, x6, 20`	`x5 = x6 | 20`	Bit-by-bit OR reg. with constant
	Exclusive or immediate	`xori x5, x6, 20`	`x5 = x6 ^ 20`	Bit-by-bit XOR reg. with constant
Shift	Shift left logical	`sll x5, x6, x7`	`x5 = x6 << x7`	Shift left by register
	Shift right logical	`srl x5, x6, x7`	`x5 = x6 >> x7`	Shift right by register
	Shift right arithmetic	`sra x5, x6, x7`	`x5 = x6 >> x7`	Arithmetic shift right by register
	Shift left logical immediate	`slli x5, x6, 3`	`x5 = x6 << 3`	Shift left by immediate
	Shift right logical immediate	`srli x5, x6, 3`	`x5 = x6 >> 3`	Shift right by immediate
	Shift right arithmetic immediate	`srai x5, x6, 3`	`x5 = x6 >> 3`	Arithmetic shift right by immediate

FIGURE 2.1 RISC-V assembly language revealed in this chapter. This information is also found in Column 1 of the RISC-V Reference Data Card at the front of this book.

- The rule for translating each assembly language instruction into machine language.

- A section describing each addressing mode your processor will use, explain how immediate bits are manipulated by each instruction type. be defined (i.e. is branch PC relative?).
- An explanation of any procedure call conventions, especially relating to register and stack use. You should write a minimum working example of procedure calling in your ISA.
- Example assembly language program demonstrating that your instruction set supports a program to find relative primes using the algorithm on the project page.

RISC-V Done Above

- Assembly language fragments for common operations. For example, this might be loading an address into a register, iteration, conditional statements, reading data from the input port, reading from the input port, and writing to the output port.
- Machine language translations of your assembly programs (relprime and your fragments). This code should be appropriately commented, and formatted such that the addresses and machine translations of each instruction is easily seen:

Address	Assembly	Machine code	Comments
0x0000	add x1, x2, x3	0101 1010 1111 0000	//an add instruction
0x0002	L: beq x1, x8, L	0101 1010 1111 0000	//a branch with a label

As an example of a professional report on a processor you can reference the data sheet for the MIPS R4300i processor. This has much more information than you will have ready during this milestone, but you should reference it early and often.

2. An individual design journal for each member. This must:
    - Include details about what that member worked on for the week (including time taken)
    - Include design decisions the team and individual member made during this milestone, including summary of meetings' outcomes, and work that may not be reflected in the design journal.
    - Include a list of tasks the member was assigned for the next milestone (planning for the next Milestone), including estimated time required for each task.
    - Be commited by the individual author. Each member must commit their own log. Teammates cannot do this for you.

- If you need help writing the journal, try adding one or two sentences for each work session in the form "I did X because of Y". The justification ("Y") is really important to show the depth of your design process.