Department of Electrical and Computer Engineering
Queen's University

ELEC-374 Digital Systems Engineering
Laboratory Project

Winter 2017

Designing a Simple RISC Computer

Objective: The purpose of this project is to design, simulate, implement, and verify a Simple RISC Computer (Mini SRC) consisting of a simple RISC processor, memory, and I/O. You are to use the Altera Quartus II design software for this purpose. The system is to be implemented on the Altera Cyclone III chip (EP3C16F484) of the Altera DE0 evaluation board.

The Mini SRC is similar to the SRC described in the Lab Notes (available on the course website), reproduced from the text by Heuring and Jordan. The Datapath, Control unit, and Memory Interface for Mini SRC have the same relationship as shown in Figure 4.1 on page 142 of the Lab Notes for the SRC system. The processor design is given by the information contained in Figures and Tables on pages 143 through 167 of the Lab Notes.

The Mini SRC is a 32-bit machine, having a 32-bit datapath, with sixteen 32-bit registers R0 to R15, with R15 acting as the stack pointer SP, R14 acting as the return address register RA (holding the return address for a jal instruction), R10 to R13 as the argument registers A0 to A3, and R8 and R9 as the return value registers V0 and V1. It also has two dedicated 32-bit registers HI and LO for multiplication and division instructions. Note that Mini SRC does not have a condition code register. Rather, it allows any of the general-purpose registers to hold a value to be tested for conditional branching.

The instructions are one word (32-bit) long each. The *Arithmetic Logic Unit* (ALU) performs 12 operations: addition, subtraction, multiplication, division, shift right, shift left, rotate right, rotate left, logical AND, logical OR, Negate (2's complement), and NOT (1's complement).

The instructions in the Mini SRC can be categorized as Load and Store Instructions, Arithmetic and Logical instructions, Conditional Branch instructions, Jump instructions, Input/Output instructions, and miscellaneous instructions. There are no push and pop instructions (they can be implemented by other instructions). The following addressing modes are supported: Direct, Index, Register Indirect, Immediate, Relative, and Inherent. The following is a formal definition of the Mini SRC.

Processor State

PC<31..0>:	32-bit Program Counter (PC)
IR<31..0>:	32-bit Instruction Register (IR)
R[0..15]<31..0>:	Sixteen 32-bit registers named R[0] through R[15]
R[15]<31..0>:	Stack Pointer (SP)
R[14]<31..0]:	Return Address Register (RA)
R[10..13]<31..0>:	Four Argument Registers, named A[0] through A[3]
R[8..9]<31..0>:	Two Return Value Registers, named V[0] and V[1]
HI<31..0>:	32-bit HI Register dedicated to keep the high-order word of a Multiplication product, or the Remainder of a Division operation
LO<31..0>:	32-bit LO Register dedicated to keep the low-order word of a Multiplication product, or the Quotient of a Division operation

Memory State

Mem[0..511]<31..0>:	512 words (32 bits per word) of memory
MDR<31..0>:	32-bit memory data register
MAR<31..0>:	32-bit memory address register

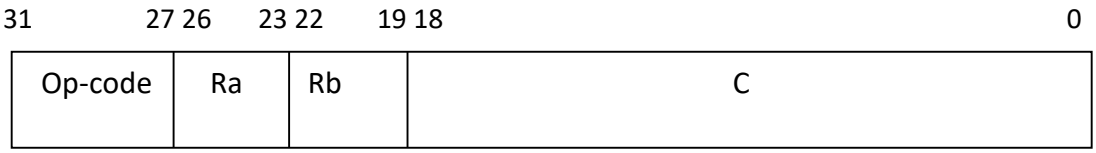
I/O State

In.Port<31..0>:	32-bit input port
Out.Port<31..0>:	32-bit output port
Run.Out:	Run/halt indicator
Stop.In:	Stop signal
Reset.In:	Reset signal

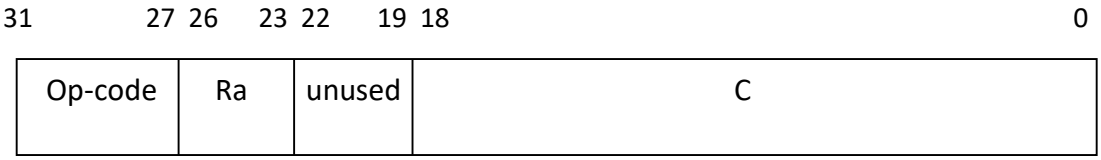
Instruction formats:

1. Load and Store instructions: operands in memory can be accessed only through load/store instructions.

   (a) ld, ldi, st

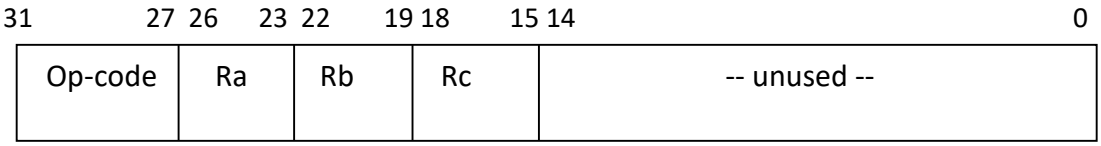


   (b) ldr, str

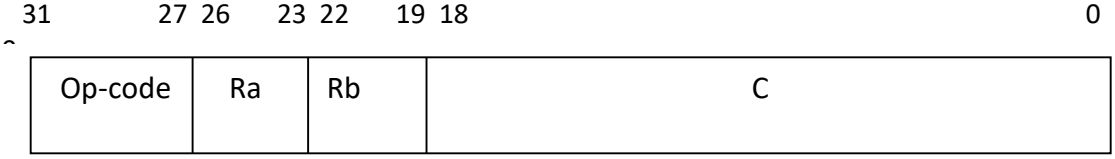


2. Arithmetic and Logical instructions:

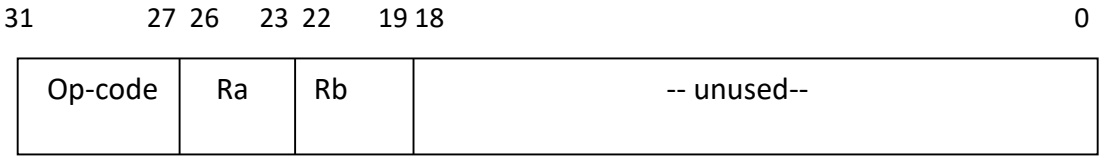
   (a) add, sub, and, or, shr, shl, ror, rol



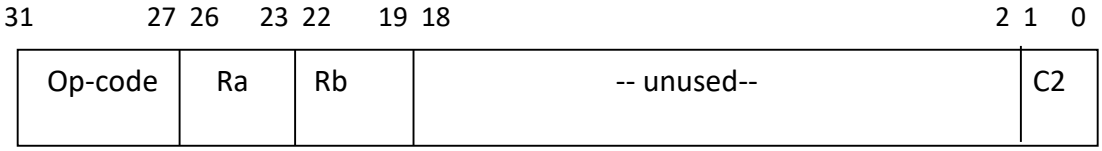
   (b) addi, andi, ori



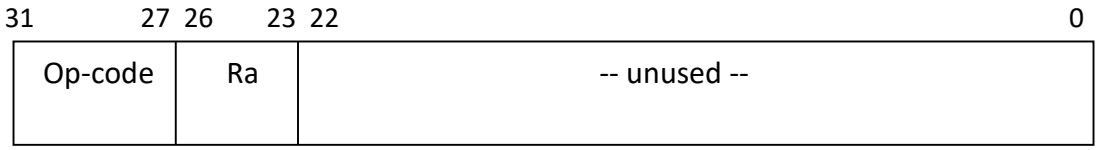
(c) mul, div, neg, not



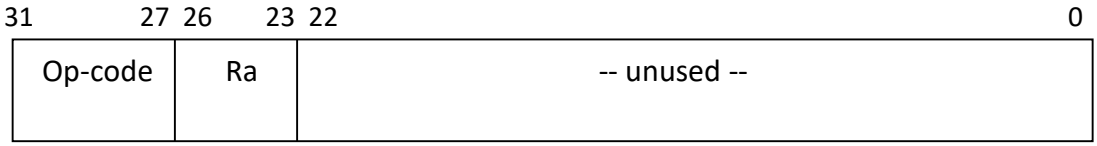
3. Branch instructions: brzr, brnz, brmi, brpl



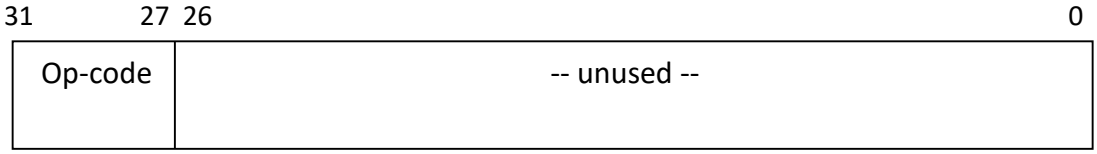
4. Jump instructions: jr, jal



5. Input/Output and MFHI/MFLO instructions: in, out, mfhi, mflo



6. Miscellaneous instructions: nop, halt



Op-code:	specifies the operation to be performed.
Ra, Rb, Rc:	0000: R0, 0001: R1, …, 1111: R15
C:	constant (data or address)
C2:	condition (00: branch if zero, 01: branch if nonzero, 10: branch if positive, 11: branch if negative)

Instructions:

The instructions (with their op-code patterns shown in parentheses) perform the following operations:
Notation: x: 0 or 1 - : unused

Load and Store Instructions

1(a): ld, ldi, st

		Assembly language	
Load direct (00000xxxx0000xxxxxxxxxxxxxxxxxxx)	R[Ra] ← M[C (sign-extended)] Direct addressing, Rb = R0	ld	Ra, C
Load indexed (00000xxxxxxxxxxxxxxxxxxxxxxxxxxx)	R[Ra] ← M[R[Rb] + C (sign-extended)] Indexed addressing, Rb ≠ R0 If C = 0 → Register Indirect addressing	ld	Ra, C(Rb)
Load immediate (00001xxxx0000xxxxxxxxxxxxxxxxxxx)	R[Ra] ← C (sign-extended) Immediate addressing, Rb = R0	ldi	Ra, C
(00001xxxxxxxxxxxxxxxxxxxxxxxxxxx)	R[Ra] ← R[Rb] + C (sign-extended) Immediate addressing, Rb ≠ R0 If C = 0 → instruction acts like a simple register transfer If C ≠ 0 and Ra = Rb → Increment/decrement instruction	ldi	Ra, C(Rb)
Store direct (00010xxxx0000xxxxxxxxxxxxxxxxxxx)	M[C (sign-extended)] ← R[Ra] Direct addressing, Rb = R0	st	C, Ra
Store indexed (00010xxxxxxxxxxxxxxxxxxxxxxxxxxx)	M[R[Rb] + C (sign-extended)] ← R[Ra] Indexed addressing, Rb ≠ R0 If C = 0 → Register Indirect addressing	st	C(Rb), Ra

1(b): ldr, str

Load relative (00011xxxx----xxxxxxxxxxxxxxxxxxx)	R[Ra] ← M[PC + C (sign-extended)] Relative addressing	ldr	Ra, C
Store relative (00100xxxx----xxxxxxxxxxxxxxxxxxx)	M[PC + C (sign-extended)] ← R[Ra] Relative addressing	str	C, Ra

Arithmetic and Logical Instructions

2(a): add, sub, and, or, shr, shl, ror, rol

Add (00101xxxxxxxxxxx----------)	R[Ra] ← R[Rb] + R[Rc]	add	Ra, Rb, Rc

Sub (00110xxxxxxxxxxx----------)	R[Ra] ← R[Rb] - R[Rc]	sub	Ra, Rb, Rc
AND (00111xxxxxxxxxxx----------)	R[Ra] ← R[Rb] ∧ R[Rc]	and	Ra, Rb, Rc
OR (01000xxxxxxxxxxx----------)	R[Ra] ← R[Rb] $_{v}$ R[Rc]	or	Ra, Rb, Rc
Shift right (01001xxxxxxxxxxx----------)	Shift R[Rb] right into R[Ra] by count in R[Rc]	shr	Ra, Rb, Rc
Shift left (01010xxxxxxxxxxx----------)	Shift R[Rb] left into R[Ra] by count in R[Rc]	shl	Ra, Rb, Rc
Rotate right (01011xxxxxxxxxxx----------)	Rotate R[Rb] right into R[Ra] by count in R[Rc]	ror	Ra, Rb, Rc
Rotate left (01100xxxxxxxxxxx----------)	Rotate R[Rb] left into R[Ra] by count in R[Rc]	rol	Ra, Rb, Rc

2(b): addi, andi, ori

Add immediate (01101xxxxxxxxxxxxxxxxxxxxxxxxxx)	R[Ra] ← R[Rb] + C (sign-extended) Immediate addressing If C = 0 ➔ instruction acts like a simple register transfer If C ≠ 0 and Ra = Rb ➔ Increment/decrement instruction Similar to ldi, however Rb can be any register.	addi	Ra, Rb, C
AND immediate (01110xxxxxxxxxxxxxxxxxxxxxxxxxx)	R[Ra] ← R[Rb] ∧ C (sign-extended) Immediate addressing	andi	Ra, Rb, C
OR immediate (01111xxxxxxxxxxxxxxxxxxxxxxxxxx)	R[Ra] ← R[Rb] $_{v}$ C (sign-extended) Immediate addressing	ori	Ra, Rb, C

2(c): mul, div, neg, not

Multiply (10000xxxxxxxx----------)	HI, LO ← R[Ra] × R[Rb]	mul	Ra, Rb
Divide (10001xxxxxxxx----------)	HI, LO ← R[Ra] ÷ R[Rb]	div	Ra, Rb

Negate (10010xxxxxxxx----------)	R[Ra] ← - R[Rb]	neg	Ra, Rb
NOT (10011xxxxxxxx----------)	R[Ra] ← $\overline{R[Rb]}$	not	Ra, Rb

Conditional Branch Instructions

brzr, brnz, brmi, brpl

Branch (10100xxxxxxxx----------xx)	PC ← R[Rb]	if R[Ra] meets the condition		
	"branch if zero"	C2 = 00	brzr	Ra, Rb
	"branch if nonzero"	C2 = 01	brnz	Ra, Rb
	"branch if positive"	C2 = 10	brpl	Ra, Rb
	"branch if negative"	C2 = 11	brmi	Ra, Rb

Jump Instructions

jr, jal

jr (10101xxxx----------)	PC ← R[Ra] If Ra = R14, it is for procedure return	jr	Ra
jal (10110xxxx----------)	R[14] ← PC + 4 PC ← R[Ra]	jal	Ra

Input/Output and MFHI/MFLO Instructions

in, out, mfhi, mflo

Input (10111xxxx----------)	R[Ra] ← In.Port	in	Ra
Output (11000xxxx----------)	Out.Port ← R[Ra]	out	Ra
Move from HI (11001xxxx----------)	R[Ra] ← HI	mfhi	Ra
Move from LO (11010xxxx----------)	R[Ra] ← LO	mflo	Ra

Miscellaneous Instructions

nop, halt

No-operation (11011----------)	Do nothing	nop
Halt (11100----------)	Halt the control stepping process	halt

Procedure:

There are four design phases in this project, which will be tackled in 10 3-hour lab sessions. However, you are strongly advised to put sufficient time to work on your project. You may use a mixed-mode Schematic/HDL design approach for your CPU, or you may opt to use an all HDL design. It is not recommended to use a mixed VHDL/Verilog design methodology.

Phase 1: The Datapath will be partially designed and tested using Functional Simulation. Phase one is worth 7% of the course mark.

Phase 2: The Datapath will be complemented by adding the "Select and Encode logic", "CON FF Logic", branch and jump instructions, "Memory Subsystem", and the "Input/Output Ports". It will be tested using Functional Simulation. Phase two is worth 6% of the course mark.

Phase 3: The Control Unit will be designed in VHDL or Verilog, and tested using Functional Simulation. You will be provided with a simple test program to verify your Control Unit. Phase three is worth 5% of the course mark.

Phase 4: The Datapath and Control Unit together will be tested using both Functional Simulation and implementation on the Altera Cyclone III chip (EP3C16F484) of the Altera DE0 evaluation board. You will be provided with a program to test your CPU. Phase four is worth 7% of the course mark; 3% for the simulation and chip demo, and 4% for the final CPU design project report.

You should demo and hand in a mini report to your lab grad TA for phase one, two, and three consisting of Schematic Capture screen printout, testbenches and Functional Simulation results, and VHDL/verilog code. You should demo your final design in phase four, and email a comprehensive final report in pdf format detailing your design, performance results, discussions, and conclusions to both grad TAs and instructor. Deadline for each phase is shown in the following table. If you need help outside the lab, feel free to contact your TA or ask your question on the course website forum.

Schedule

	Deadline*	Automatic extension by one week	25% Penalty	40% Penalty
Phase 1 demo and report	Week 5 and 6	Week 7	Week 8	Week 9–Week 12
Phase 2 demo and report	Week 8 and 9	Week 10	Week 11	Week 12
Phase 3 demo and report	Week 10 and Week 11	Week 12	-	-
Phase 4 demo	Week 12	-	-	-
Final report	Apr 9th	-	-	-

*Note: Demo slot is during your lab period. Report deadline is at 11:59 PM of your lab day. For example, if your deadline is on Week 7 and your lab is on Tuesday, then your deadline is considered as 11:59 PM on Tuesday of Week 7.

Please note that our minimum goal is a one-bus architecture for the Mini SRC. However, there will be up to 5% **bonus** mark if you design and implement, for instance, a 3-bus architecture, new instructions, interrupt handling, VHDL/Verilog implementations of advanced techniques for ALU operations, as well as other advanced techniques such as pipelining, branch prediction, hazard detection, and superscalar design to improve performance. I would recommend everyone to tackle the 1-bus architecture first, and then aim for other improvements.