Computer Architecture

CA2

RISC-V single cycle

Mohammad Moshiri Balasoor - 810101518

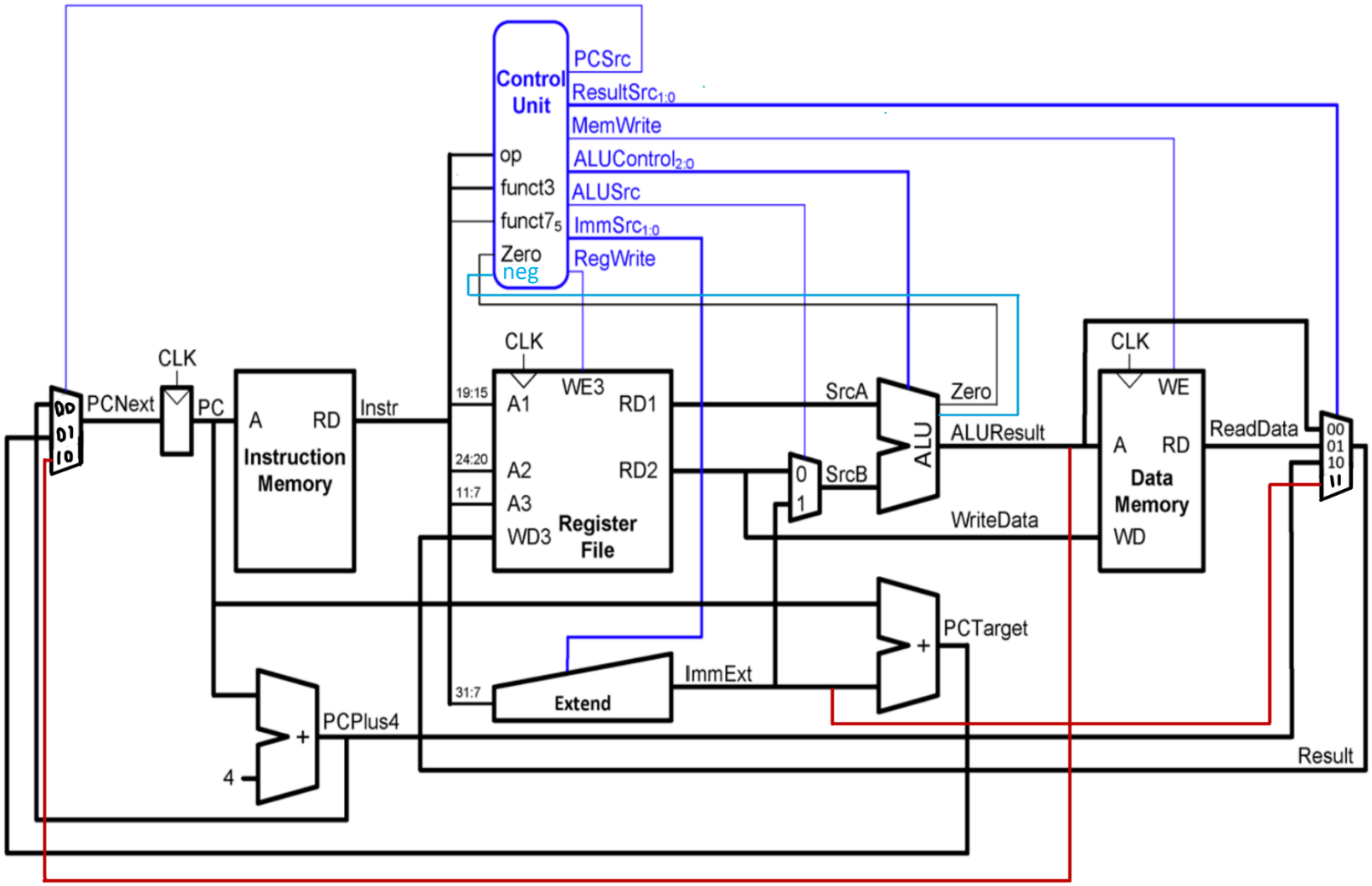
Narges Elyasi - 810100258

1403.02.19

Table 3 : RV32I RISC-V Integer instructions

op	Func3	Func7	Type	Mnemonic	Description	Operation
0000011(3)	000		I	lb rd, imm(rs1)	Load byte	rd = SignExt([Address]$_{7:0}$)
0000011(3)	001		I	lh rd, imm(rs1)	Load half	rd = SignExt([Address]$_{15:0}$)
0000011(3)	010		I	lw rd, imm(rs1)	Load word	rd = ([Address]$_{31:0}$)
0000011(3)	100		I	lbu rd, imm(rs1)	Load byte unsigned	rd = ZeroExt([Address]$_{7:0}$)
0000011(3)	101		I	lhu rd, imm(rs1)	Load half unsigned	rd = ZeroExt([Address]$_{15:0}$)
0010011(19)	000		I	addi rd, rs1, imm	ADD immediate	rd = rs1 + SignExt(imm)
0010011(19)	001		I	slli rd, rs1, uimm	Shift left logical immediate	rd = rs1 << uimm
0010011(19)	010		I	slti rd, rs1, imm	Set less than immediate	rd = rs1 < SignExt(imm)
0010011(19)	011	0000000	I	sltiu rd, rs1, imm	Set less than imm. unsigned	rd = rs1 < SignExt(imm)
0010011(19)	100		I	xori rd, rs1, imm	XOR immediate	rd = rs1 ^ SignExt(imm)
0010011(19)	101	0000000	I	srli rd, rs1, uimm	Shift right logical immediate	rd = rs1 >> uimm
0010011(19)	101	0100000	I	srai rd, rs1, uimm	Shift right arithmetic immediate	rd = rs1 >> uimm
0010011(19)	110		I	ori rd, rs1, uimm	OR immediate	rd = rs1 | SignExt(imm)
0010011(19)	111		I	andi rd, rs1, uimm	AND immediate	rd = rs1 & SignExt(imm)
0010111(23)			U	auipc rd, rs1, uimm	ADD upper immediate to PC	rd = (upimm, 12'b0) + PC
0100011(35)			S	sb rs2,imm(rs1)	Store byte	[Address]$_{7:0}$ = rs2$_{7:0}$
0100011(35)			S	sh rs2,imm(rs1)	Store half	[Address]$_{15:0}$ = rs2$_{15:0}$
0100011(35)			S	sw rs2,imm(rs1)	Store word	[Address]$_{31:0}$ = rs2
0110011(51)	000	0000000	R	add rd, rs1, rs2	ADD	rd = rs1 + rs2
0110011(51)	000	0100000	R	sub rd, rs1, rs2	SUB	rd = rs1 - rs2
0110011(51)	001	0000000	R	sll rd, rs1, rs2	Shift left logical	rd = rs1 << rs2$_{4:0}$
0110011(51)	010	0000000	R	slt rd, rs1, rs2	Set less than	rd = rs1 < rs2
0110011(51)	011	0000000	R	sltu rd, rs1, rs2	Set less than unsigned	rd = rs1 < rs2
0110011(51)	100	0000000	R	xor rd, rs1, rs2	XOR	rd = rs1 ^ rs2
0110011(51)	101	0000000	R	srl rd, rs1, rs2	Shift right logical	rd = rs1 >> rs2$_{4:0}$
0110011(51)	101	0100000	R	sra rd, rs1, rs2	Shift right arithmetic	rd = rs1 >>>rs2$_{4:0}$
0110011(51)	110	0000000	R	or rd, rs1, rs2	OR	rd = rs1 | rs2
0110011(51)	111	0000000	R	and rd, rs1, rs2	AND	rd = rs1 & rs2
0110111(55)	-		U	lui rd, upimm	Load upper immediate	rd = {upimm, 12'b0}
1100011(99)	000		B	beq rs1,rs2, label	Branch if equal =	if (rs1 == rs2) PC = BTA
1100011(99)	001		B	bne rs1,rs2, label	Branch if not equal ≠	if (rs1 != rs2) PC = BTA
1100011(99)	010		B	blt rs1,rs2, label	Branch if lower than <	if (rs1 < rs2) PC = BTA
1100011(99)	011		B	bge rs1,rs2, label	Branch if greater / equal ≥	if (rs1 ≥ rs2) PC = BTA
1100011(99)	100		B	bltu rs1,rs2, label	Branch if lower than unsigned <	if (rs1 < rs2) PC = BTA
1100011(99)	101		B	bgeu rs1,rs2, label	Branch if greater / equal unsign. ≥	if (rs1 ≥ rs2) PC = BTA
1100111(103)	000		I	jalr rd, rs1, label	Jump and link register	PC = rs1 + SignExt(imm) rd = PC + 4
1101111(111)	-		J	jal rd, label	Jump and link	PC = JTA rd = PC + 4

Datapath


Control Unit
PCSrc
ResultSrc1:0
MemWrite
ALUControl2:0
ALUSrc
ImmSrc1:0
RegWrite
op
funct3
funct75
Zero
neg
CLK
PCNext
PC
Instr
A
RD
Instruction Memory
19:15
24:20
11:7
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
CLK
WE
A
RD
Data Memory
WD
ReadData
00
01
10
11
0
1
PCTarget
+
ImmExt
31:7
Extend
PCPlus4
4
Result

Controller

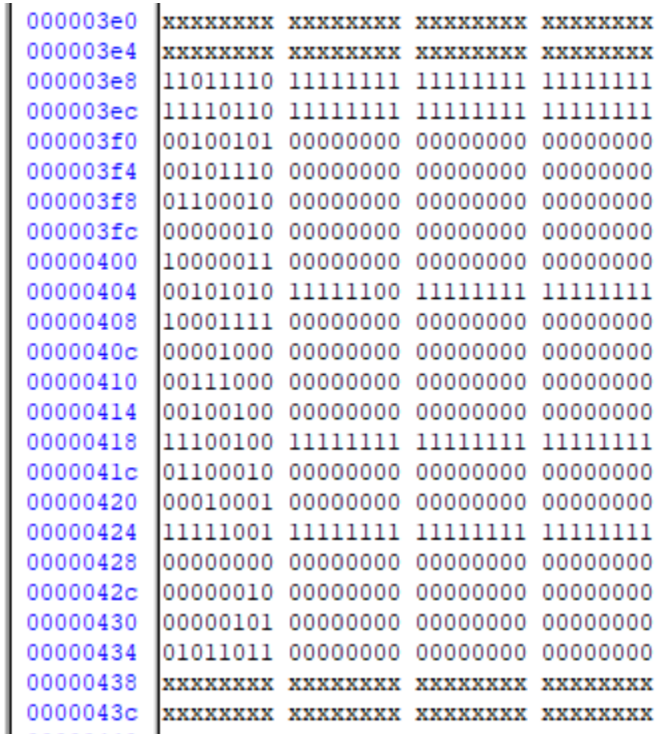
Instruction Type	Instruction	PCsrc	Result src	Mem write	ALU op	ALU control	ALU src	imm src	Reg write	Branch	jal	jalr
R-Type	add	00	00	0	10	000	0	XXX	1	0	0	0
	sub	00	00	0	10	001	0	XXX	1	0	0	0
	and	00	00	0	10	010	0	XXX	1	0	0	0
	or	00	00	0	10	011	0	XXX	1	0	0	0
	slt	00	00	0	10	101	0	XXX	1	0	0	0
	sltu	00	00	0	10	110	0	XXX	1	0	0	0
I-Type	lw	00	01	0	00	000	1	000	1	0	0	0
	addi	00	00	0	11	000	1	000	1	0	0	0
	xori	00	00	0	11	100	1	000	1	0	0	0
	ori	00	00	0	11	011	1	000	1	0	0	0
	slti	00	00	0	11	101	1	000	1	0	0	0
	sltiu	00	00	0	11	110	1	000	1	0	0	0
	jalr	10	10	0	11	000	1	000	1	0	0	1
S-Type	sw	00	XX	1	00	000	1	001	0	0	0	0
J-Type	jal	01	10	0	XX	000	X	010	1	0	1	0
B-Type	beq	**	XX	0	01	001	0	011	0	1	0	0
	bne	**	XX	0	01	001	0	011	0	1	0	0
	blt	**	XX	0	01	001	0	011	0	1	0	0
	bge	**	XX	0	01	001	0	011	0	1	0	0
U-Type	lui	00	11	0	XX	000	X	100	1	0	0	0

** PCsrc for B-Type instruction : IF branch_resault == 1 THEN PCsrc = 01 ELSE PCsrc = 00

Test:

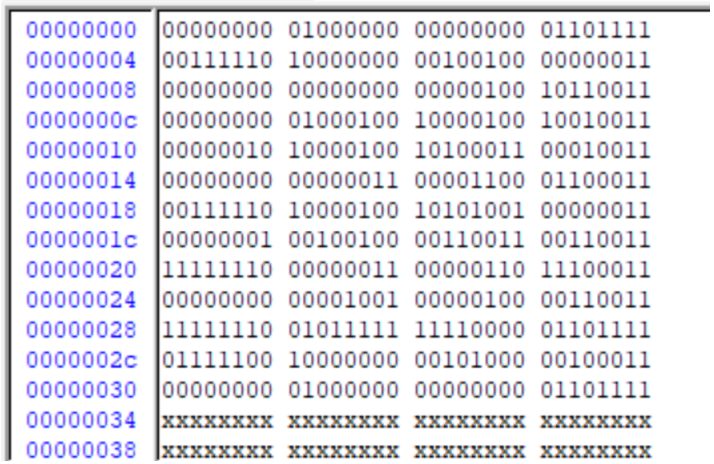
```
nums = [-34, -10, 37, 46, 98, 2, 131, -982, 143, 8, 56, 36, -28, 98, 17, -7, 0, 2, 5, 91]
```

Data Memory:



000003e0	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
000003e4	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
000003e8	11011110 11111111 11111111 11111111
000003ec	11110110 11111111 11111111 11111111
000003f0	00100101 00000000 00000000 00000000
000003f4	00101110 00000000 00000000 00000000
000003f8	01100010 00000000 00000000 00000000
000003fc	00000010 00000000 00000000 00000000
00000400	10000011 00000000 00000000 00000000
00000404	00101010 11111100 11111111 11111111
00000408	10001111 00000000 00000000 00000000
0000040c	00001000 00000000 00000000 00000000
00000410	00111000 00000000 00000000 00000000
00000414	00100100 00000000 00000000 00000000
00000418	11100100 11111111 11111111 11111111
0000041c	01100010 00000000 00000000 00000000
00000420	00010001 00000000 00000000 00000000
00000424	11111001 11111111 11111111 11111111
00000428	00000000 00000000 00000000 00000000
0000042c	00000010 00000000 00000000 00000000
00000430	00000101 00000000 00000000 00000000
00000434	01011011 00000000 00000000 00000000
00000438	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000043c	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Instruction Memory (for unsigned):



00000000	00000000 01000000 00000000 01101111
00000004	00111110 10000000 00100100 00000011
00000008	00000000 00000000 00000100 10110011
0000000c	00000000 01000100 10000100 10010011
00000010	00000010 10000100 10100011 00010011
00000014	00000000 00000011 00001100 01100011
00000018	00111110 10000100 10101001 00000011
0000001c	00000001 00100100 00110011 00110011
00000020	11111110 00000011 00000110 11100011
00000024	00000000 00001001 00000100 00110011
00000028	11111110 01011111 11110000 01101111
0000002c	01111100 10000000 00101000 00100011
00000030	00000000 01000000 00000000 01101111
00000034	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000038	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Signed numbers:

Assembly code:

```
.global _boot

.text
_boot:
    jal x0, FindMax

    FindMax:
        lw   x8, 1000(x0)          #* maxElement = mem[1000]
        add  x9, x0, x0            #* i

        Loop:
            addi x9, x9, 4         #* i += 4
            slti x6, x9, 40        #* check if 10 elements are traversed (40 = 4 * 10)
            beq  x6, x0, EndLoop   #* if 10 elements are traversed, jump to EndLoop
            lw   x18, 1000(x9)     #* element = mem[i]
            slt  x6, x8, x18       #* check if element is greater than maxElement
            beq  x6, x0, Loop      #* if element is not greater than maxElement, jump to Loop
            add  x8, x18, x0       #* maxElement = element
            jal  x0,Loop           #* jump to Loop

        EndLoop:
            sw x8, 2000(x0)        #* mem[2000] = maxElement
            jal x0,End             #* return

    End:
```

Registr file:

00000000	00000000000000000000000000000000
00000001	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000002	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000003	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000004	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000005	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000006	00000000000000000000000000000000
00000007	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000008	00000000000000000000000010001111
00000009	00000000000000000000000001010000
0000000a	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000b	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000011	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000012	00000000000000000000000000001000
00000013	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000014	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000015	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000017	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000018	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001a	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001b	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Unsigned numbers:

Assembly code:

```
.global _boot

.text
_boot:
    jal x0, FindMax

    FindMax:
        lw   x8, 1000(x0)          #* maxElement = mem[1000]
        add  x9, x0, x0            #* i

        Loop:
            addi x9, x9, 4         #* i += 4
            slti x6, x9, 40        #* check if 10 elements are traversed (40 = 4 * 10)
            beq  x6, x0, EndLoop   #* if 10 elements are traversed, jump to EndLoop
            lw   x18, 1000(x9)     #* element = mem[i]
            sltu  x6, x8, x18       #* check if element is greater than maxElement
            beq  x6, x0, Loop      #* if element is not greater than maxElement, jump to Loop
            add  x8, x18, x0       #* maxElement = element
            jal  x0,Loop           #* jump to Loop

        EndLoop:
            sw x8, 2000(x0)        #* mem[2000] = maxElement
            jal x0,End             #* return

    End:
```

Registr file: