Computer Architecture

CA3

RISC-V multi cycle

Mohammad Moshiri Balasoor - 810101518

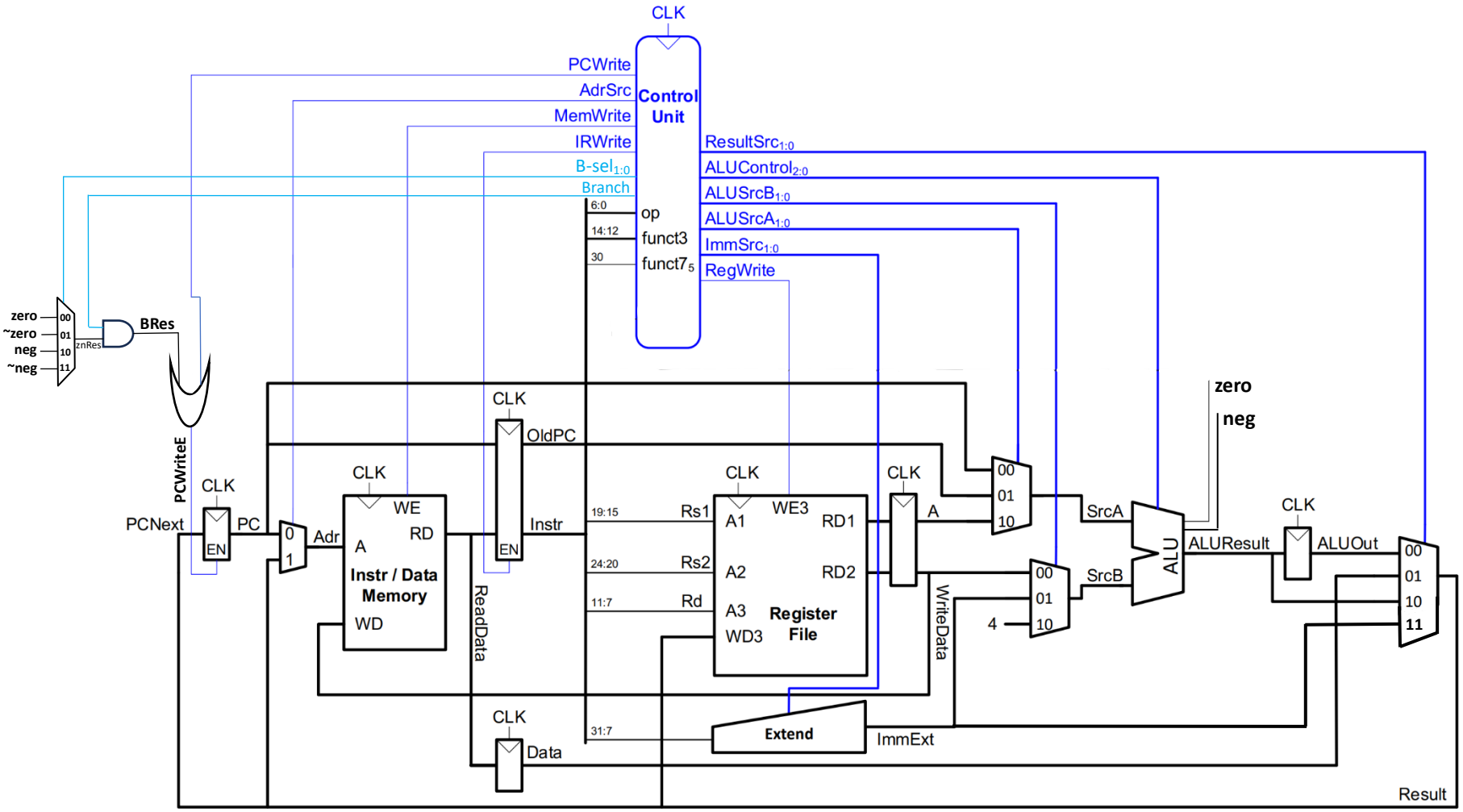
Narges Elyasi - 810100258

1403.03.05

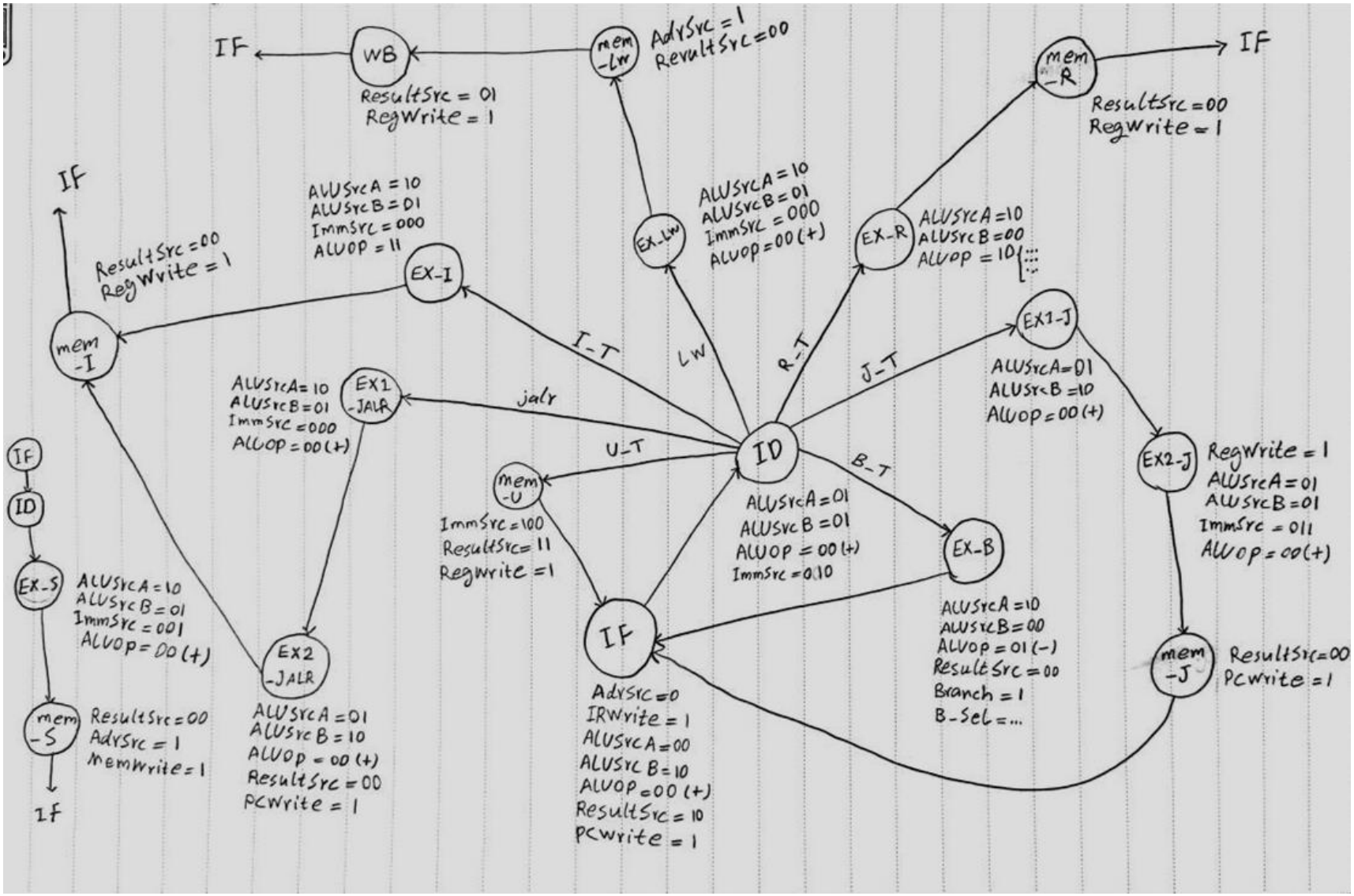
Table 3 : RV32I RISC-V Integer instructions

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


CLK
PCWrite
AdrSrc
MemWrite
IRWrite
B-sel1:0
Branch
Control Unit
ResultSrc1:0
ALUControl2:0
ALUSrcB1:0
ALUSrcA1:0
ImmSrc1:0
RegWrite
6:0
op
14:12
funct3
30
funct75
zero
~zero
neg
~neg
00
01
10
11
znRes
BRes
PCWriteE
CLK
PCNext
PC
EN
0
1
Adr
WE
RD
A
Instr / Data Memory
WD
OldPC
Instr
ReadData
19:15
Rs1
24:20
Rs2
11:7
Rd
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
A
WriteData
4
SrcA
SrcB
ALU
zero
neg
ALUResult
ALUOut
31:7
Extend
ImmExt
Data
Result

Controller

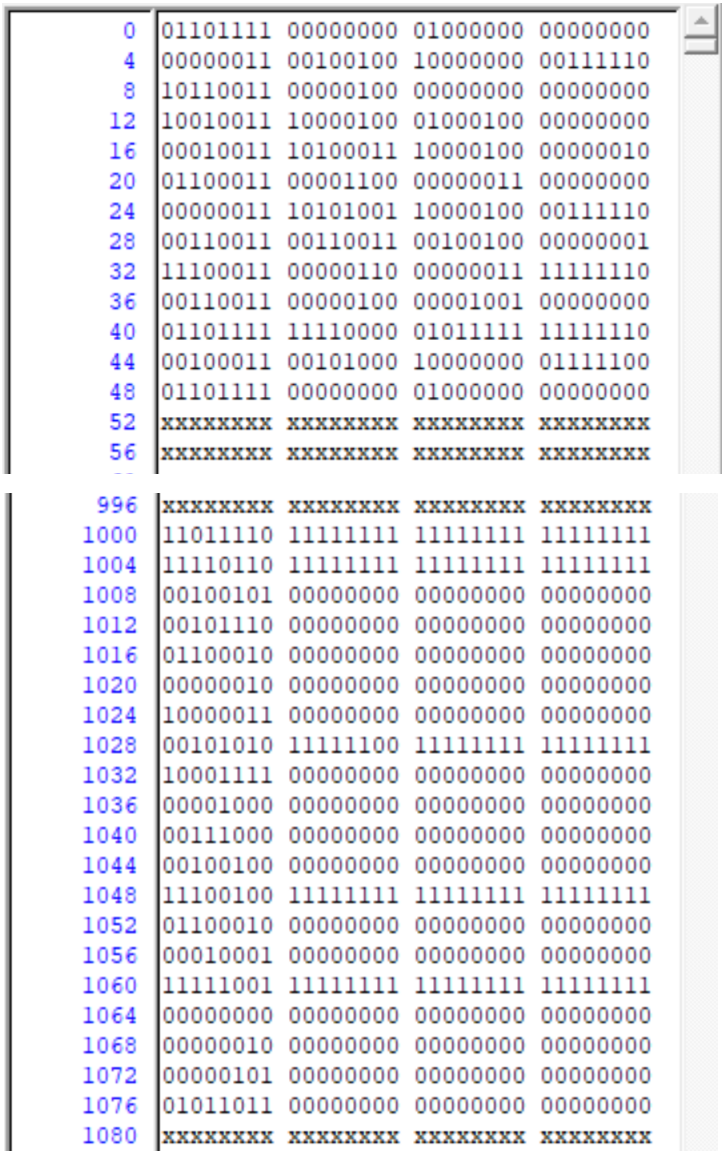

IF
WB
ResultSrc = 01
RegWrite = 1
mem -Lw
AdrSrc = 1
ResultSrc = 00
IF
mem -R
ResultSrc = 00
RegWrite = 1
IF
ResultSrc = 00
RegWrite = 1
mem -I
ALUSrcA = 10
ALUSrcB = 01
ImmSrc = 000
ALUOP = 11
EX_I
EX_Lw
ALUSrcA = 10
ALUSrcB = 01
ImmSrc = 000
ALUOP = 00 (+)
EX_R
ALUSrcA = 10
ALUSrcB = 00
ALUOP = 10
I_T
LW
R_T
J_T
EX1 -JALR
ALUSrcA = 10
ALUSrcB = 01
ImmSrc = 000
ALUOP = 00 (+)
jalr
EX1_J
ALUSrcA = 01
ALUSrcB = 10
ALUOP = 00 (+)
IF
ID
U_T
B_T
mem -U
ImmSrc = 100
ResultSrc = 11
RegWrite = 1
ID
ALUSrcA = 01
ALUSrcB = 01
ALUOP = 00 (+)
ImmSrc = 010
EX2_J
RegWrite = 1
ALUSrcA = 01
ALUSrcB = 01
ImmSrc = 011
ALUOP = 00 (+)
EX_S
ALUSrcA = 10
ALUSrcB = 01
ImmSrc = 001
ALUOP = 00 (+)
EX_B
ALUSrcA = 10
ALUSrcB = 00
ALUOP = 01 (-)
ResultSrc = 00
Branch = 1
B-Sel = ...
EX2 -JALR
ALUSrcA = 01
ALUSrcB = 10
ALUOP = 00 (+)
ResultSrc = 00
PCWrite = 1
IF
AdrSrc = 0
IRWrite = 1
ALUSrcA = 00
ALUSrcB = 10
ALUOP = 00 (+)
ResultSrc = 10
PCWrite = 1
mem -J
ResultSrc = 00
PCWrite = 1
mem -S
ResultSrc = 00
AdrSrc = 1
MemWrite = 1
If

Test

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

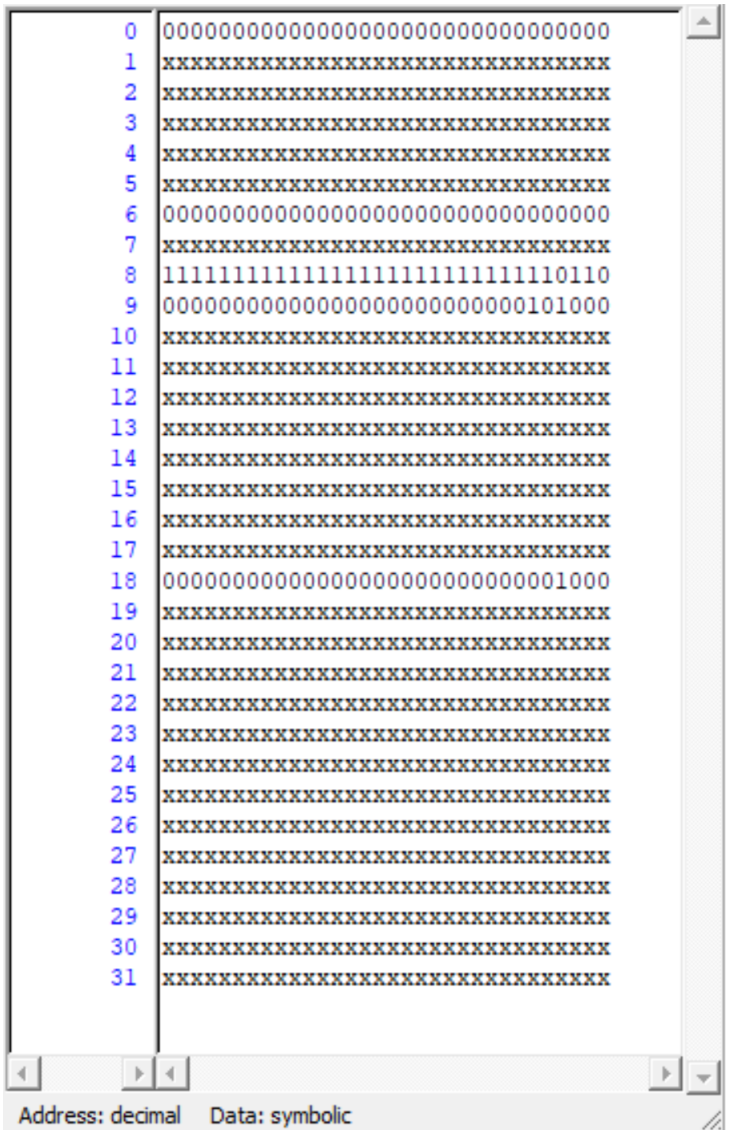
```
.global _boot

.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:

```

Instr / Data Memory :



0	01101111 00000000 01000000 00000000
4	00000011 00100100 10000000 00111110
8	10110011 00000100 00000000 00000000
12	10010011 10000100 01000100 00000000
16	00010011 10100011 10000100 00000010
20	01100011 00001100 00000011 00000000
24	00000011 10101001 10000100 00111110
28	00110011 00110011 00100100 00000001
32	11100011 00000110 00000011 11111110
36	00110011 00000100 00001001 00000000
40	01101111 11110000 01011111 11111110
44	00100011 00101000 10000000 01111100
48	01101111 00000000 01000000 00000000
52	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
56	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
996	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
1000	11011110 11111111 11111111 11111111
1004	11110110 11111111 11111111 11111111
1008	00100101 00000000 00000000 00000000
1012	00101110 00000000 00000000 00000000
1016	01100010 00000000 00000000 00000000
1020	00000010 00000000 00000000 00000000
1024	10000011 00000000 00000000 00000000
1028	00101010 11111100 11111111 11111111
1032	10001111 00000000 00000000 00000000
1036	00001000 00000000 00000000 00000000
1040	00111000 00000000 00000000 00000000
1044	00100100 00000000 00000000 00000000
1048	11100100 11111111 11111111 11111111
1052	01100010 00000000 00000000 00000000
1056	00010001 00000000 00000000 00000000
1060	11111001 11111111 11111111 11111111
1064	00000000 00000000 00000000 00000000
1068	00000010 00000000 00000000 00000000
1072	00000101 00000000 00000000 00000000
1076	01011011 00000000 00000000 00000000
1080	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Register File :



0	00000000000000000000000000000000
1	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
6	00000000000000000000000000000000
7	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
8	11111111111111111111111111110110
9	00000000000000000000000000101000
10	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
11	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
12	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
13	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
14	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
15	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
16	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
17	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
18	00000000000000000000000000001000
19	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
20	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
21	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
22	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
23	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
24	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
25	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
26	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
27	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
28	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
29	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
30	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
31	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Address: decimal Data: symbolic