MIPS Instructions

R-type:

			Func:
add	R1, R2, R3	R1 = R2 + R3	100000
sub	R1, R2, R3	R1 = R2 - R3	100010
slt	R1, R2, R3	R1 = (R2 < R3) ? 1 : 0	101010
and	R1, R2, R3	R1 = R2 & R3	100100
or	R1, R2, R3	R1 = R2 | R3	100101

Machine:

opcode[6]	sr1[5]	sr2[5]	dr[5]	shift[5]	func[6]

31 26 25 21 20 16 15 11 10 6 5 0

l-type:

			Opcode:
addi	R1, R2, Num	R1 = R2 + Num	001000
stli	R1, R2, Num	R1 = (R2 < Num) ? 1 : 0	001010

opcode[6]	sr1[5]	dr[5]	imm[16]

31 26 25 21 20 16 15 0

Mem-type:

			Opcode:
lw	R1, Num(R2)	R1 = Mem[R2 + Num]	100011
sw	R1, Num(R2)	Mem[R2 + Num] = R1	101011

Machine:

opcode[6]	sr1[5]	sr2[5]	imm[16]

31 26 25 21 20 16 15 0

Jump1:

		Opcode:
j Adr	PC = {(PC+4)[31:28], Adr << 2}	000010
jal Adr	j to Adr and R31 = PC + 4	000011

Machine:

opcode[6]	adr[26]
31 26	25 0

Jump2:

		Opcode:
jr R1	PC = R1	111111

Machine:

opcode[6]	sr1[5]	—
31 26	25 21	20 0

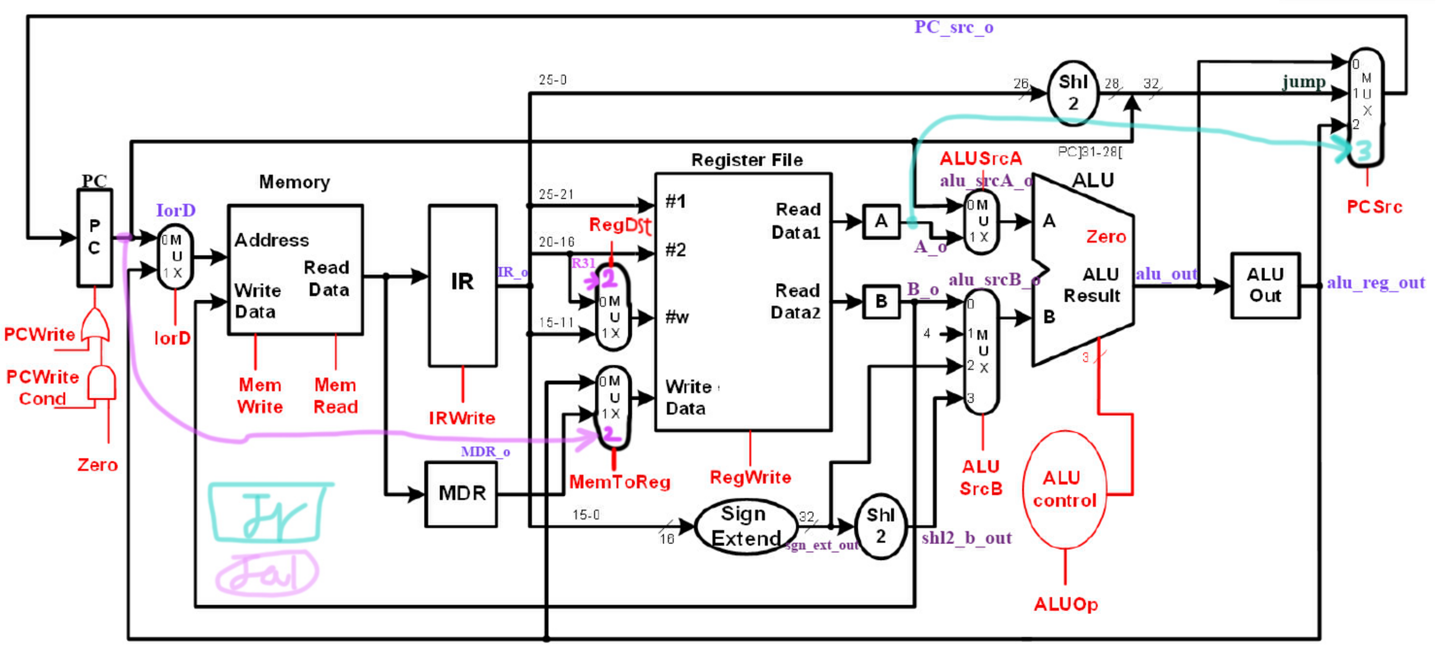
Branch:

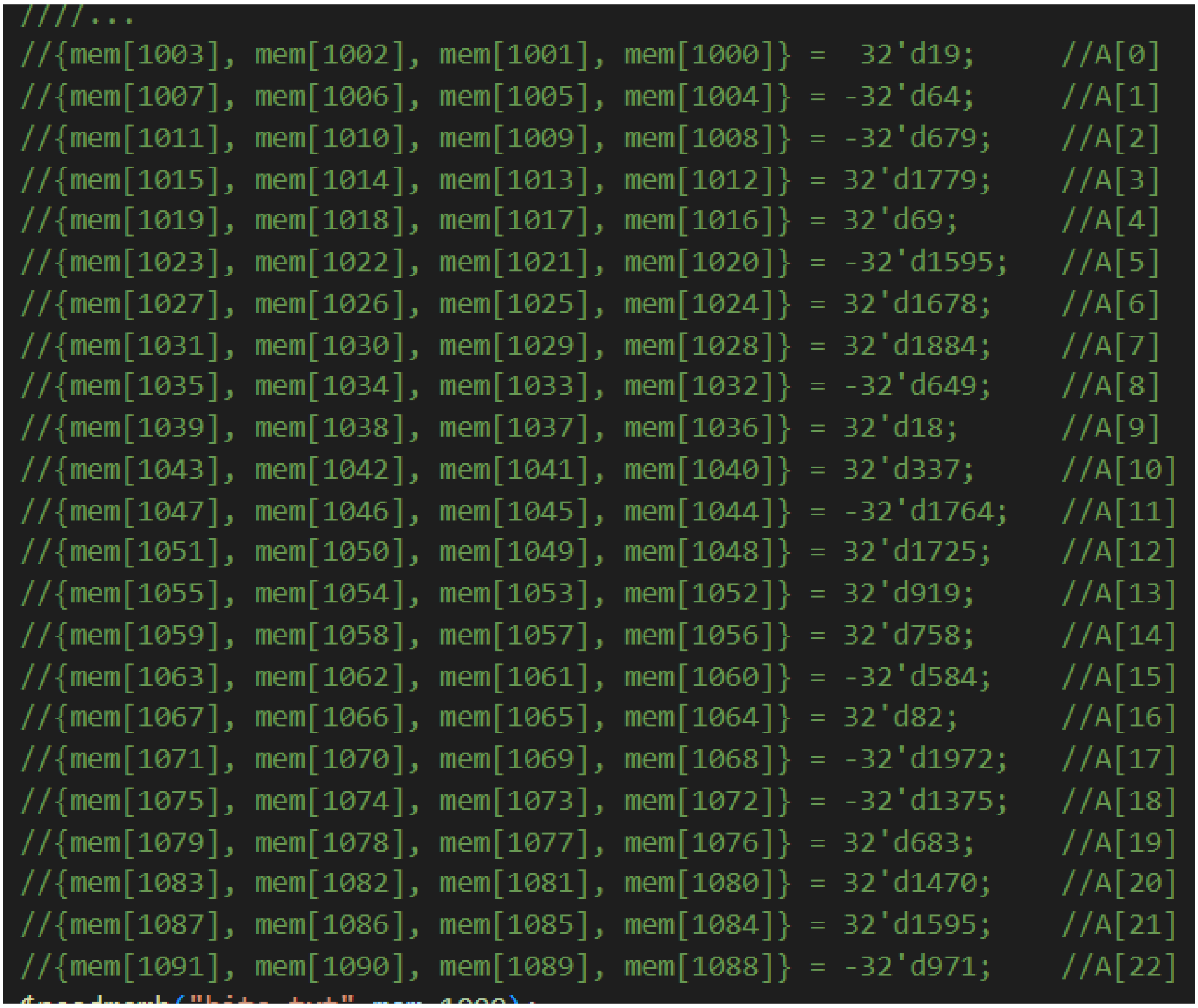
		Opcode:
beq R1, R2, Adr	PC = (R1==R2) ? (PC + 4 + Adr<<2) : PC+4 ;	000100

Machine:

opcode[6]	sr2[5]	sr1[5]	adr[16]
31 26	25 21	20 16	15 0

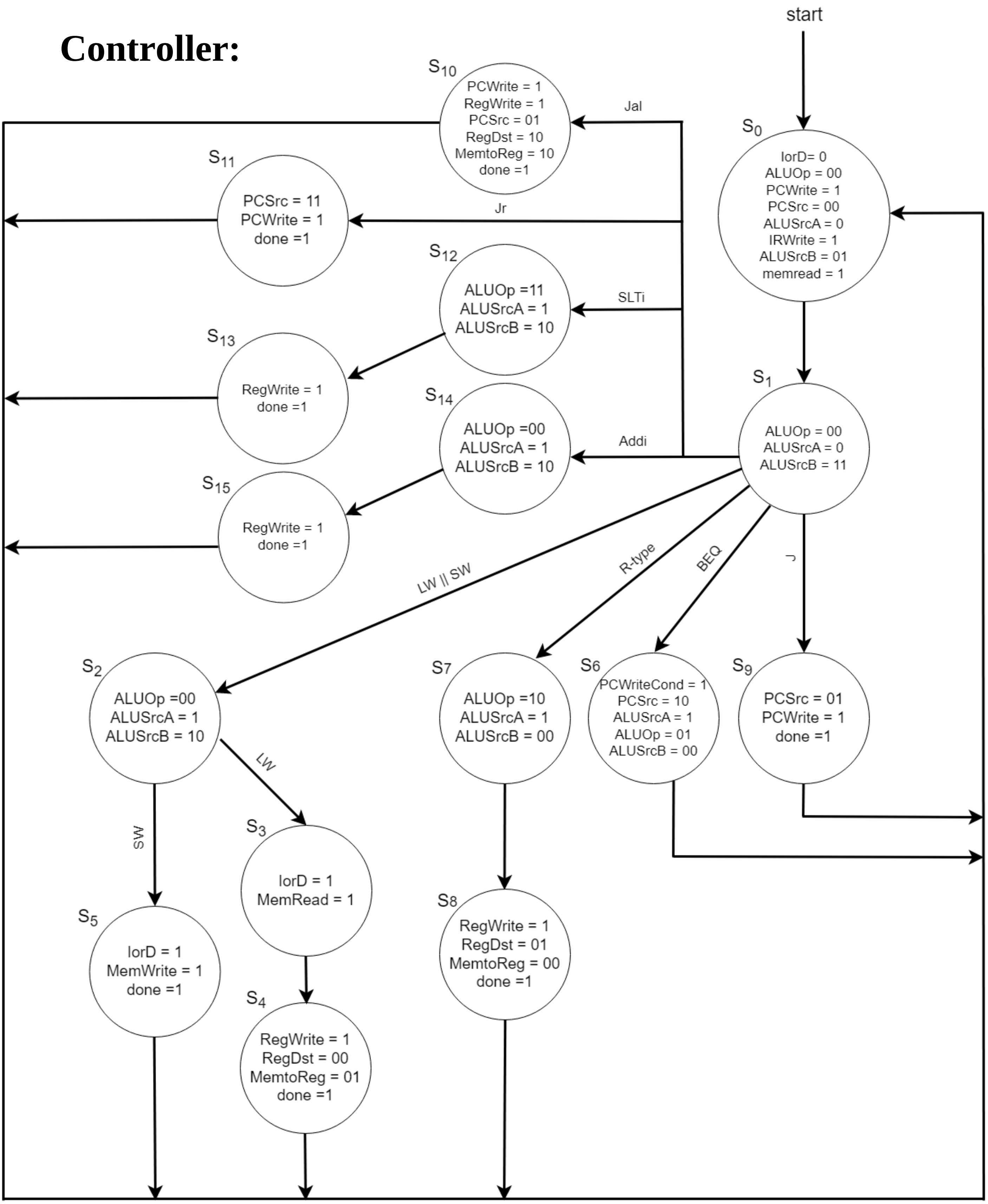
DataPath:

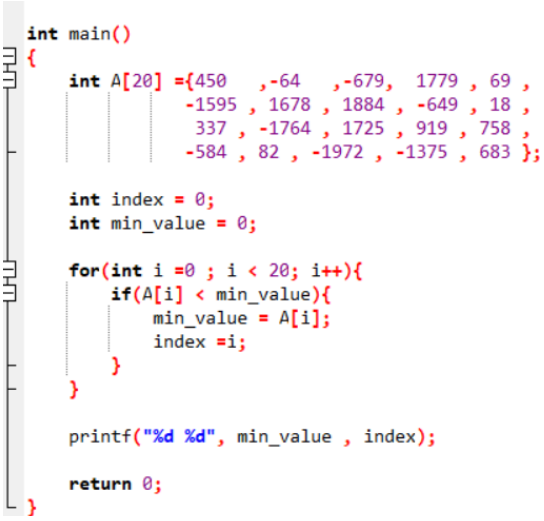




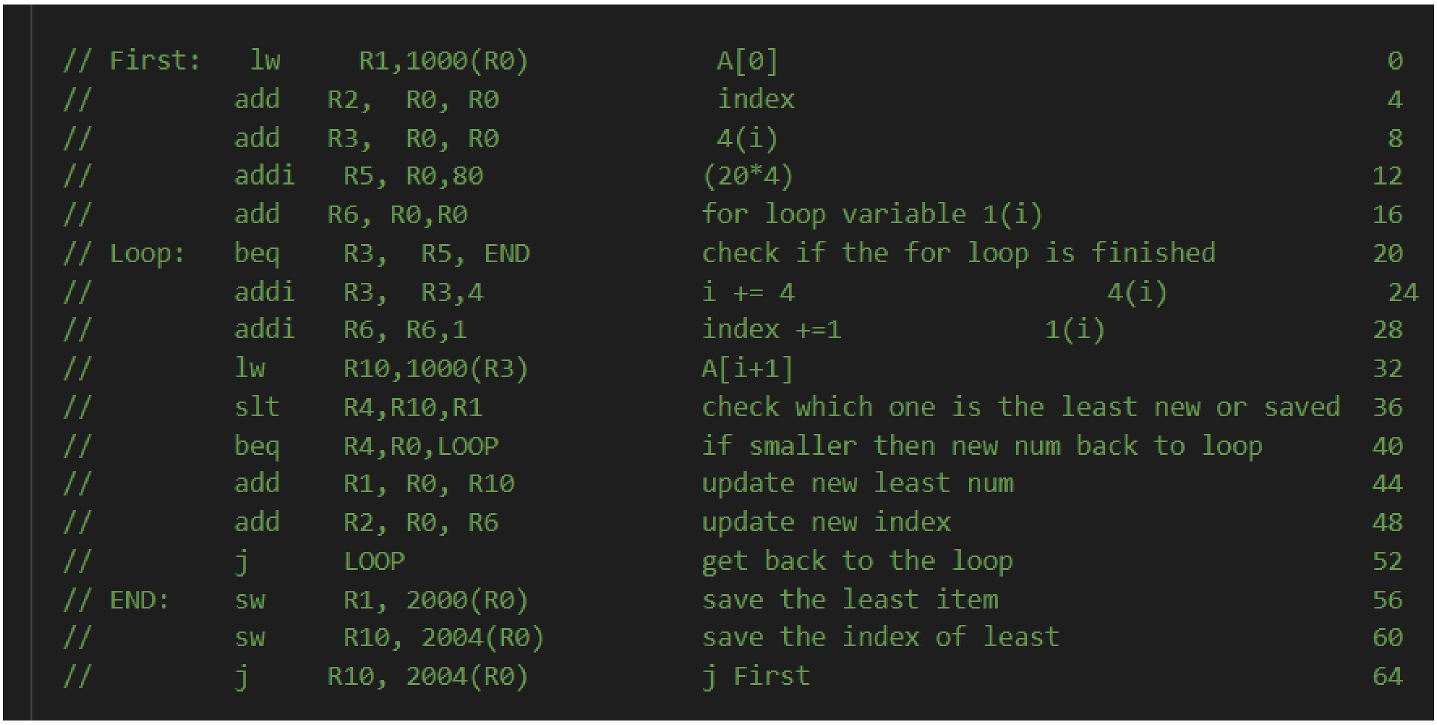
```
////...
//{mem[1003], mem[1002], mem[1001], mem[1000]} =  32'd19;     //A[0]
//{mem[1007], mem[1006], mem[1005], mem[1004]} = -32'd64;     //A[1]
//{mem[1011], mem[1010], mem[1009], mem[1008]} = -32'd679;    //A[2]
//{mem[1015], mem[1014], mem[1013], mem[1012]} = 32'd1779;    //A[3]
//{mem[1019], mem[1018], mem[1017], mem[1016]} = 32'd69;      //A[4]
//{mem[1023], mem[1022], mem[1021], mem[1020]} = -32'd1595;   //A[5]
//{mem[1027], mem[1026], mem[1025], mem[1024]} = 32'd1678;    //A[6]
//{mem[1031], mem[1030], mem[1029], mem[1028]} = 32'd1884;    //A[7]
//{mem[1035], mem[1034], mem[1033], mem[1032]} = -32'd649;    //A[8]
//{mem[1039], mem[1038], mem[1037], mem[1036]} = 32'd18;      //A[9]
//{mem[1043], mem[1042], mem[1041], mem[1040]} = 32'd337;     //A[10]
//{mem[1047], mem[1046], mem[1045], mem[1044]} = -32'd1764;   //A[11]
//{mem[1051], mem[1050], mem[1049], mem[1048]} = 32'd1725;    //A[12]
//{mem[1055], mem[1054], mem[1053], mem[1052]} = 32'd919;     //A[13]
//{mem[1059], mem[1058], mem[1057], mem[1056]} = 32'd758;     //A[14]
//{mem[1063], mem[1062], mem[1061], mem[1060]} = -32'd584;    //A[15]
//{mem[1067], mem[1066], mem[1065], mem[1064]} = 32'd82;      //A[16]
//{mem[1071], mem[1070], mem[1069], mem[1068]} = -32'd1972;   //A[17]
//{mem[1075], mem[1074], mem[1073], mem[1072]} = -32'd1375;   //A[18]
//{mem[1079], mem[1078], mem[1077], mem[1076]} = 32'd683;     //A[19]
//{mem[1083], mem[1082], mem[1081], mem[1080]} = 32'd1470;    //A[20]
//{mem[1087], mem[1086], mem[1085], mem[1084]} = 32'd1595;    //A[21]
//{mem[1091], mem[1090], mem[1089], mem[1088]} = -32'd971;    //A[22]
$readmemh("hits.txt", mem,1000);
```

Controller:

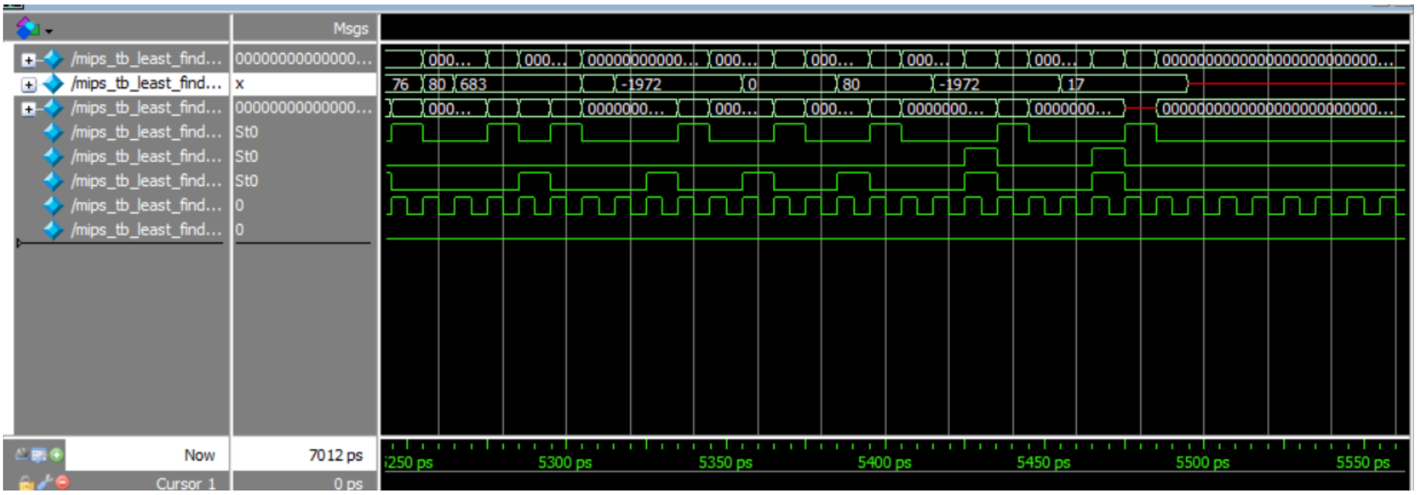

start
S0
IorD= 0
ALUOp = 00
PCWrite = 1
PCSrc = 00
ALUSrcA = 0
IRWrite = 1
ALUSrcB = 01
memread = 1
S1
ALUOp = 00
ALUSrcA = 0
ALUSrcB = 11
Jal
S10
PCWrite = 1
RegWrite = 1
PCSrc = 01
RegDst = 10
MemtoReg = 10
done =1
Jr
S11
PCSrc = 11
PCWrite = 1
done =1
SLTi
S12
ALUOp =11
ALUSrcA = 1
ALUSrcB = 10
S13
RegWrite = 1
done =1
Addi
S14
ALUOp =00
ALUSrcA = 1
ALUSrcB = 10
S15
RegWrite = 1
done =1
LW || SW
R-type
BEQ
J
S2
ALUOp =00
ALUSrcA = 1
ALUSrcB = 10
LW
SW
S3
IorD = 1
MemRead = 1
S4
RegWrite = 1
RegDst = 00
MemtoReg = 01
done =1
S5
IorD = 1
MemWrite = 1
done =1
S7
ALUOp =10
ALUSrcA = 1
ALUSrcB = 00
S8
RegWrite = 1
RegDst = 01
MemtoReg = 00
done =1
S6
PCWriteCond = 1
PCSrc = 10
ALUSrcA = 1
ALUOp = 01
ALUSrcB = 00
S9
PCSrc = 01
PCWrite = 1
done =1



```c
int main()
{
    int A[20] ={450   ,-64   ,-679,  1779 , 69 ,
                -1595 , 1678 , 1884 , -649 , 18 ,
                 337 , -1764 , 1725 , 919 , 758 ,
                -584 , 82 , -1972 , -1375 , 683 };

    int index = 0;
    int min_value = 0;

    for(int i =0 ; i < 20; i++){
        if(A[i] < min_value){
            min_value = A[i];
            index =i;
        }
    }

    printf("%d %d", min_value , index);

    return 0;
}
```



```
// First:  lw     R1,1000(R0)          A[0]                                0
//        add    R2,  R0, R0           index                               4
//        add    R3,  R0, R0           4(i)                                8
//        addi    R5, R0,80           (20*4)                              12
//        add    R6, R0,R0            for loop variable 1(i)              16
// Loop:  beq     R3,  R5, END        check if the for loop is finished   20
//        addi    R3,  R3,4           i += 4               4(i)            24
//        addi    R6, R6,1            index +=1          1(i)             28
//        lw      R10,1000(R3)        A[i+1]                              32
//        slt     R4,R10,R1           check which one is the least new or saved  36
//        beq     R4,R0,LOOP          if smaller then new num back to loop  40
//        add     R1, R0, R10         update new least num                44
//        add     R2, R0, R6          update new index                    48
//        j       LOOP                get back to the loop                52
// END:   sw      R1, 2000(R0)        save the least item                 56
//        sw      R10, 2004(R0)       save the index of least             60
//        j      R10, 2004(R0)        j First                             64
```



همانطور که در مقادیر می بینید بیست خانه اول(از A[0] تا A[19]) را چک می کند و در ارایه ۱۷ ام مقدار -۱۹۷۲ که کوچکترین مقدار است را ذخیره می کند.