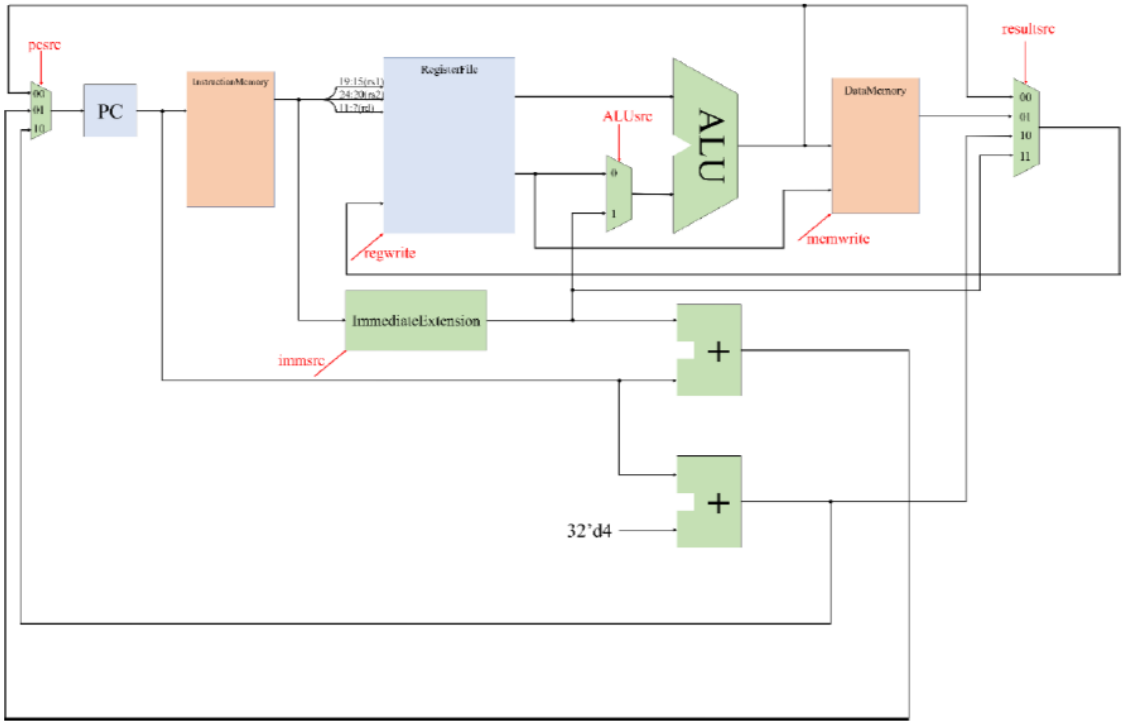
حامد گل کار 810101499

پریسا محمدی 810101509

دیتاپث جدید به شکل زیر خواهد بود که به دیتاپث اولیه دستورات LUI ,JALr اضافه شده است



و واحد کنترل مسئول انتخاب مسیر درست در این دیتاپث است.

پیاده سازی این سیستم و واحد کنترل در فایل های همراه آمده است و ریز مسائل در آنجا حل شده است و هنگام تحویل توضیح داده خواهد شد.

پس از پیاده سازی این پردازنده به تست آن میپردازیم.
کد اسمبلی که برای تست آن نوشتیم به صورت زیر است.

```
1   lw s0, 0x5b0(zero)
2   add s1, zero, zero
3   Loop:
4   addi s1, s1, 4
5   slti t1, s1, 80
6   beq t1, zero, EndLoop
7   lw s2, 0x5b0(s1)
8   slt t1, s0, s2
9   beq t1, zero, Loop
10  add s0, s2, zero
11
12  jal zero, Loop
13  EndLoop:
14  sw s0, 0x5ac(zero)
15  Trap:
16  jal zero, Trap
```

که در آن داریم:

Register	Purpose	Register Number (X_i)
S_2	Reading data from memory (temp)	X_{18}
S_1	Saving i	X_9
S_0	Saving maximum	X_8
t_1	Saving SLT comparisons' results	X_6

با استفاده از جدول زیر کد اسمبلی را به کد ماشین تبدیل میکنیم.

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011 51	0x0 0	0x00 0	rd = rs1 + rs2	
sub	SUB	R	0110011 51	0x0 0	0x20 32	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011 51	0x6 6	0x00 0	rd = rs1 | rs2	
and	AND	R	0110011 51	0x7 7	0x00 0	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011 51	0x2 2	0x00 0	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011 51	0x3 3	0x00 0	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011 19	0x0 0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011 19	0x4 4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011 19	0x6 6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011 19	0x2 2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011 19	0x3 3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011 3	0x2 2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011 35	0x2 2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011 99	0x0 0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011 99	0x1 1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011 99	0x4 4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011 99	0x5 5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111 111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111 103	0x0 0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111 55			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

حال کد ماشین به دست آمده را به صورت دو بایت دوبایت (همانطور که در دیتاپث مشخص کردیم) مینویسیم و در ممووری نیز اعداد رندومی قرار میدهیم تا بزرگترین آنها را پیدا کرده و در رجیستر فایل بنویسد.

نتیجه شبیه سازی به صورت زیر است که نشان از صحت عملکرد دارد.