# Mustafa Aydoğan 191101002 BİL361 Ödev-1 Raporu

### 1. Evrişim İşleminde Kullanılacak Buyruklar

**a. ADD** add rd, rs1, rs2

**b. SUB** sub rd, rs1, rs2

c. ADDI addi rd, rs1, #imm

**d. BEQ** beq rs1, rs2, #imm

e. SLL sll rd, rs1, rs2

**f. LW** lw rd, #imm(rs1)

g. SW sw rs2, #imm(rs1)

**h. JALR** jalr rd, rs1, #imm

#### 2. Buyrukların Türleri ve İçerikleri

-	ADD:												
31		25 24		20	19	15	5 14	12	11	7 6		0	
	funct7		rs2		rs	:1	fu	nct3	rd		opcode		
	7		5			5		3	5		7		
	0000000		src2		sr	c1	ADD/S	LT/SLT	U dest		OP		
	0000000	)		rs2	?	r	s1	000	1	rd	0110011		Al
			+					- 00		-	1		

SUB:							
1 25	24	20 19	15 14	12 11	7 6	0	
funct7	rs2	rs	1 fu	nct3	rd	opcode	
7	5	5		3	5	7	_
0000000	src2	src	1 ADD/S	SLT/SLTU	dest	OP	
0000000	src2	src	1 AND/0	OR/XOR	dest	OP	
0000000	src2	src	1 SLI	/SRL	dest	OP	
0100000	src2	src	e1 SUE	S/SRA	dest	OP	
0100000	]	rs2	rs1	000	$\operatorname{rd}$	0110011	St
	funct7 7 0000000 0000000 0000000 0100000	funct7         rs2           7         5           00000000         src2           0000000         src2           0000000         src2           0100000         src2	funct7         rs2         rs           7         5         5           00000000         src2         src           00000000         src2         src           00000000         src2         src           01000000         src2         src	funct7         rs2         rs1         funct           7         5         5           00000000         src2         src1         ADD/S           00000000         src2         src1         AND/G           00000000         src2         src1         SLL           0100000         src2         src1         SUE	funct7         rs2         rs1         funct3           7         5         5         3           00000000         src2         src1         ADD/SLT/SLTU           0000000         src2         src1         AND/OR/XOR           0000000         src2         src1         SLL/SRL           0100000         src2         src1         SUB/SRA	funct7         rs2         rs1         funct3         rd           7         5         5         3         5           00000000         src2         src1         ADD/SLT/SLTU         dest           0000000         src2         src1         AND/OR/XOR         dest           0000000         src2         src1         SLL/SRL         dest           0100000         src2         src1         SUB/SRA         dest	funct7         rs2         rs1         funct3         rd         opcode           7         5         5         3         5         7           00000000         src2         src1         ADD/SLT/SLTU         dest         OP           0000000         src2         src1         AND/OR/XOR         dest         OP           0000000         src2         src1         SLL/SRL         dest         OP           0100000         src2         src1         SUB/SRA         dest         OP

	-	ADDI:										
	31		20 19		15 14		12	11	7 6		0	
		imm[11:0]		rs1		funct3	П	rd		opcode		
'		12		5		3		5		7		
		0		0		ADDI		0		OP-IMM		
		imm[11:0]		1	rs1	000		rd	-	0010011		ADDI

## - BEQ:

31 30 25	5 24 20 19	15 14 1:	2 11 8	3 7	6 0	
imm[12] $imm[10:5]$	rs2 rs1	funct3	imm[4:1]	imm[11]	opcode	]
1 6	5 5	3	4	1	7	-
offset $[12 10:5]$	src2 src	1 BEQ/BNE	offset[1]	1 4:1]	BRANCH	
imm[12 10:5]	rs2	rs1	000 in	nm[4:1 11]	1100011	$_{ m BEQ}$

## - SLL:

31	25	24	2	0 19	15	5 14	12	11	7.6	3	0
funct	7		rs2	rs	s <b>1</b>	fur	ict3	rd		opcode	
7			5		5		3	5		7	
00000	000	1	src2	sr	c1	ADD/S	LT/SLTU	J dest	;	OP	
00000	000	1	src2	sr	c1	AND/C	R/XOR	dest	;	OP	
00000	000	1	src2	sr	c1	SLL	/SRL	dest	;	OP	
01000	000	1	src2	sr	c1	SUB	/SRA	dest	;	OP	
0000	0000		rs	32	r	rs1	001		$\operatorname{rd}$	0110011	SLL

### - **LW:**

	31	20 19	15 14	12 11	7 6	0	
	imm[11:0]	rs1	fu	nct3 r	d	opcode	
	12	5		3	5	7	_
	offset[11:0]	base	e wi	dth de	est	LOAD	
	imm[11:0]	r	s1	010	rd	0000011	LW
L	11111[11.0]	1,	51	010	Tu	0000011	LV

## - **SW:**

31	25 24	20 19 15	5 14 12	11 7	6 0	
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	]
7	5	5	3	5	7	_
offset[11:5]	$\operatorname{src}$	base	width	offset[4:0]	STORE	
imm[11:5]	rs2	rs1	010	imm[4:	0] 0100011	$\int SW$

### - JALR:

31 20	19 15	14 12	11 7	6 0	
imm[11:0]	rs1	funct3	$^{\mathrm{rd}}$	opcode	]
12	5	3	5	7	
offset[11:0]	base	0	dest	JALR	
imm[11:0]	rs1	000	rd	1100111	JALR
•	•	•	•	•	

#### 3. Evrişim İşleminde Kullanılan Buyrukların Genel Amaçları:

- 1. **ADD:** ADD buyruğu R-tipi bir buyruktur. İşlem kodu 0110011'dir. Funct3 değeri 000 iken funct7 değeri 0000000'dır. Rs1 yazmacındaki değeri ve rs2 yazmacındaki değeri okuyup toplar. Sonucu rd yazmacına yazar.
- 2. **SUB**: SUB buyruğu R- tipi bir buyruktur. İşlem kodu ADD buyruğunun işlem kodu (0110011) ile aynıdır. Funct3 değeri 000 iken toplamadan farklı olarak funct7 değeri 0100000'dir. Rs1 yazmacındaki değerden rs2 yazmacındaki değeri çıkarır. Sonucu rd yazmacına yazar.
- 3. ADDI: ADDI buyruğu I- tipi bir buyruktur. İşlem kodu 0010011'dir. Funct3 değeri 000'dır. 12 bitlik anlık değeri genişletir ve rs1 yazmacındaki değer ile toplar. Sonucu rd yazmacına yazar.
- 4. **BEQ:** BEQ buyruğu B-tipi bir buyruktur. İşlem kodu 1100011'dir. Funct3 değeri 000'dır. Rs1 yazmacındaki değer ile rs2 yazmacındaki değeri karşılaştırır. Eşit olması durumunda program sayacını (PS) anlık değer kadar artırır.
- 5. **SLL**: SLL buyruğu R-tipi bir buyruktur. İşlem kodu 0110011'dir. Funct3 değeri 001 iken funct7 değeri 0000000'dır. Rs1 yazmacındaki değeri rs2 yazmacındaki değer kadar mantıksal olarak sola kaydırır. Sonucu rd yazmacına yazar.
- 6. **LW**: LW buyruğu I-tipi bir buyruktur. İşlem kodu 0000011'dir. Funct3 değeri 010'dır. Rs2 yazmacındaki değer ile anlık değeri toplayarak bellek adresi elde eder. Veri belleğine giderek elde edilen adresteki veriyi rd yazmacına yazar.
- 7. **SW**: SW buyruğu S-tipi bir buyruktur. İşlem kodu 0100011'dir. Funct3 değeri 010'dır. Rs1 yazmacındaki değer ile anlık değeri toplayarak bellek adresi elde eder. Veri belleğine giderek elde edilen adrese rs2 yazmacındaki veriyi yazar.
- 8. **JALR**: JALR buyruğu I-tipi bir buyruktur. İşlem kodu 1100111'dir. Funct3 değeri 000'dır. Program sayacının değerini rd yazmacına yazar. Rs1 yazmacındaki değer ile anlık değeri toplayarak program sayacını bu değer ile günceller. Buyruk belleğine giderek elde edilen adresteki veriyi rd yazmacına yazar.

#### 4. Buyrukların Evrişim İşleminde Kullanılma Amaçları:

- 1. ADD buyruğu evrişim işleminde yazmaçlardaki değerleri toplamak için kullanılmıştır. Ayrıca yazmaçlar sıfırlanmak istendiğinde ADD buyruğu kullanılmıştır. Sıfırıncı yazmacın kendi ile toplanması ve hedef yazmaca yazılması esnasında kullanılmıştır. Bir başka kullanım amacı da çarpım işlemidir. İki sayı çarpılmak istendiğinde BEQ buyruğu ve JALR buyrukları kullanılarak bir döngü elde edilmiştir. Bu döngü esnasında ADD buyruğu kullanılarak çarpma işlemi gerçekleştirilmiştir.
- 2. **SUB** buyruğu evrişim işleminde en az kullanılan buyruklardan biridir. Yazmaçların farkı alınıp başka bir yazmaca yazılması istendiğinde kullanılmıştır.
- 3. **ADDI** buyruğu evrişim işleminde anlık değerleri yazmaçlara ve belleğe aktarmak için kullanılmıştır.
- 4. BEQ buyruğu evrişim işleminde gerekli olan döngüyü elde etme esnasında kullanılmıştır. BEQ buyruğu ve JALR buyruğu program sayacını günceller ve kontrol eder. Bu sayede bir dallanmayı kontrol eder ve döngü elde edilir.
- 5. **SLL** buyruğu evrişim işleminde az sayıda kullanılan bir buyruktur. Kullanılma amacı veri belleği ile işlem yapılması için gerekli olan bellek adresinin elde edilmesidir. Veri belleğinden yükleme yapmak için bellek adresine ihtiyacımız vardır. Bellek adresleri 4'ün katı şeklinde olduğu için değişkenlerimizi **SLL** buyruğu sayesinde 4'ün katı haline getirebiliriz.
- 6. **LW** buyruğu veri belleğinden değer okuyup belirtilen yazmaca yazma esnasında kullanılmıştır.
- 7. **SW** buyruğu veri belleğinde değer saklamak amacıyla kullanılmıştır. Belirtilen bellek adresine giderek verileri belleğe yazar.
- JALR buyruğu evrişim işleminde gerekli olan döngüyü sağlamak için kullanılmıştır. Aynı zamanda çarpma işlemi yapılacağı zaman gerekli olan döngü bu buyruk sayesinde gerçekleşir.

#### 5. Ödev Kapsamında Assembly Dilinde Yazılan Programın Açıklaması:

- 0| addi x1, x0, 1
- 4| addi x8, x0, #G\_BOYUT
- 8 addi x9, x0, #F\_BOYUT
- 12 addi x11, x0, #G\_BASLANGIC
- 16 addi x12, x0, #F\_BASLANGIC
- 20 addi x13, x0, #H\_BASLANGIC
- 24 | addi x18, x0, 2
- 28 | sub x10, x8, x9
- 32 add x10, x10, x1
- 36 | beq x6, x10, 92
- 40 | beq x7, x9, 60
- 44 add x14, x6, x7
- 48 | sll x14, x14, x18
- 52 | add x14, x14, x11
- 56 | sll x15, x7, x18
- 60 add x15, x12, x15
- 64 | lw x3, 0(x14)
- 68 | lw x4, 0(x15)
- 72 | beq x17, x4, 16
- 76 | add x2, x2, x3
- 80 | add x17, x17, x1
- 84 | jalr x19, x0, 72
- 88 | add x17, x0, x0
- 92 add x7, x7, x1
- 96 | jalr x19, x0, 40
- 100 | sll x16, x6, x18
- 104 | add x16, x16, x13
- 108 | sw x2, 0(x16)
- 112 | add x2, x0, x0
- 116 | add x6, x6, x1
- 120 | add x7, x0, x0
- 124 | jalr x19, x0, 36

- x1 yazmacına daha sonra kullanmak için 1 değeri atandı.
- x8, x9, x11, x12 ve x13 yazmaçlarına sırası ile anlık değerler olan #G\_BOYUT, #F\_BOYUT, #G\_BASLANGIC, #F\_BASLANGIC ve #H\_BASLANGIC değerleri atandı.
- x18 yazmacına daha sonra kullanmak üzere 2 değeri anlık olarak atandı.
- x8 yazmacındaki değerden x9 yazmacındaki değer çıkarıldı ve x10 yazmacına yazıldı.
- x10 yazmacındaki değer 1 artırıldı.
- Döngü oluşturmak için dallanma buyruğu kullanıldı.
   x6 ile x10 yazmacındaki değerler karşılaştırıldı. Eğer eşitse program sayacını 92 artırdı. Eşit değilse 4 artırarak sonraki buyruğa geçti.
- İkinci bir döngü oluşturmak için BEQ buyruğu kullanıldı. x7 ve x9 değerleri karşılaştırıldı. Eşit olması durumunda program sayacı 60 artırıldı.
- SLL buyruğu kullanılarak x17 yazmacındaki değer 2 bit sola kaydırıldı. Böylece sayı 4 ile çarpılmış oldu.
- ADD komutları kullanılarak bellekten veri okumak ve yazmak için gerekli adresler elde edildi. Adresler x14 ve x15 yazmaçlarında saklanmaktadır.
- LW buyruğu ile bellekten okuma yapıldı.
- Bir döngüye daha ihtiyacımız olduğu için yine dallanma buyruğu kullanıldı.
- Bu döngüde evrişim işleminin en temel işlemi olan çarpma işlemi sağlanmış oldu.
- x17 değeri bir artırıldı.
- JALR buyruğu kullanılarak program sayacındaki değer x19 yazmacında yazıldı ve PC 72 olarak güncellendi.
- Döngü bittikten sonra x17 yazmacı sıfırlandı. x7 yazmacındaki değer 1 artırıldı.
- JALR buyruğu kullanılarak program sayacındaki değer x19 yazmacında yazıldı ve PC 40 olarak güncellendi.
- SLL buyruğu ve ADD buyruğu kullanılarak adres değeri hesaplandı.
- x2 yazmacındaki hesaplanmış değer, SW buyruğu ile x16 yazmacındaki adrese yazıldı.
- x2 ve x7 numaralı yazmaçlar sıfırlandı.
- x6 yazmacındaki değer 1 artırıldı.
- Döngü oluşması için JALR buyruğu kullanıldı ve PC 36 olarak güncellendi.

### 6. Programın Java Dilinde Yazılmış Hali:

```
1 int sum = 0;
2
3 for(int i=0; i<G_BOYUT-F_BOYUT+1; i++)
4 {
5    for(int j=0; i<F_BOYUT; j++)
6    {
7       for(int k=0; i<F[j]; k++)
8       {
9         sum = sum + G[i+j];
10       }
11    }
12
13    H[i] = sum;
14    sum = 0;
15 }</pre>
```

## 7. Değişkenlerin Saklandığı Yazmaçlar:

x0> 0	x16> H[i] bellek adresi
x1> 1	x17> k
x2> sum	x18> 2
x3> G[i+j]	x19> JALR buyruğu sonrasında Program Counter
x4> F[j]	x20> 0
x5> H[i]	x21> 0
x6>i	x22> 0
x7> j	x23> 0
x8> #G_BOYUT	x24> 0
x9> #F_ BOYUT	x25> 0
x10> #G_ BOYUT - #F_ BOYUT + 1	x26> 0
x11> #G_BASLANGIC	x27> 0
x12> #F_BASLANGIC	x28> 0
x13> #H_BASLANGIC	x29> 0
x14> G[i+j] bellek adresi	x30> 0
x15> F[j] bellek adresi	x31> 0