

## تکلیف سری اول ریزپردازنده

سارا برادران (شماره دانشجویی : ۹۶۲۴۱۹۳) – غزاله زمانی (شماره دانشجویی : ۹۷۲۸۰۴۳)

### Question 1)

- a. 11 address, 1 data SRAM → organization :  $2^{11} * 1$  , capacity : 2 Kbit
- b. 17 address, 8 data SRAM → organization :  $2^{17} * 8$  , capacity : 1 Mbit
- c. 9 address, 1 data DRAM → organization :  $2^{18} * 1$  , capacity : 256 Kbit

### Question 2)

"U.S.A. is a country in North America"

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

ASCII : 55 2E 53 2E 41 2E 20 69 73 20 61 20 63 6F 75 6E 74 72 79 20  
69 6E 20 4E 6F 72 74 68 20 41 6D 65 72 69 63 61

### Question 3)

The largest number that can be carried is  $2^{32} - 1$

### Question 4)

A14, A15 control lines

→ For activating:  $A14 \& \sim A15 = 1 \rightarrow A14 = 1, A15 = 0$

A0 – A13 address lines

→  $A0 - A13 = 0$  or  $1$

01XX XXXX XXXX XXXX → address

Address Range → 0x4000 – 0x7FFF

### Question 5)

Y0 → 0x0000 – 0x1FFF

→ 0000 0000 0000 0000 – 0001 1111 1111 1111

The size of each memory block is  $2^{13} * 8 = 64$  Kbit

	A15	A14	A13	A12	A11 ... A0
Y0	0	0	0	0 1	0x000 0xFFF
Y1	0	0	1	0 1	0x000 0xFFF
Y2	0	1	0	0 1	0x000 0xFFF
Y3	0	1	1	0 1	0x000 0xFFF
Y4	1	0	0	0 1	0x000 0xFFF
Y5	1	0	1	0 1	0x000 0xFFF
Y6	1	1	0	0 1	0x000 0xFFF
Y7	1	1	1	0 1	0x000 0xFFF

### Question 6)

a) ADD R20, R11 → True

b) ADD R16, R1 → True

c) ADD R52, R16 → False

because we have only 32 general purpose registers(R0 – R31). we don't have R52

d) LDI R16, \$255 → False

because  $\$255 = (0010\ 0101\ 0101)_2 > \$FF$  and R16 is a 8 bit register.

e) LDI R23, 0xF5 → True

### Question 7)

```
LDI R20, 0xFF
```

```
LDI R21, 1
```

```
ADD R20, R21
```

\$FF

\$01

-----

\$100 → Z = 1, C = 1

### Question 8)

a)

```
LDI R20, 0x54
```

```
LDI R25, 0xC4
```

```
ADD R20, R25
```

\$54

\$C4

-----

\$118 → C = 1

b)

```
LDI R23, 0
LDI R16, 0xFF
ADD R23, R16
```

\$00

\$FF

-----

\$FF → C = 0

c)

```
LDI R30, 0xFF
LDI R18, 0x05
ADD R30, R18
```

\$FF

\$05

-----

\$104 → C = 1

### Question 9)

R0 → 8bit

R24 → 8bit

PORTA → 8bit

Data memory Location \$300 → 8bit

Program memory Location \$300 → 16bit

### Question 10)

R0 → (largest unsigned value :  $2^8 - 1 = 255$ )  
(largest signed value :  $2^7 - 1 = 127$ )

R24 → (largest unsigned value :  $2^8 - 1 = 255$ )  
(largest signed value :  $2^7 - 1 = 127$ )

PORTA → (largest unsigned value :  $2^8 - 1 = 255$ )  
(largest signed value :  $2^7 - 1 = 127$ )

Data memory Location \$300 →

(largest unsigned value :  $2^8 - 1 = 255$ )

(largest signed value :  $2^7 - 1 = 127$ )

Program memory Location \$300 →  $2^{16} - 1$

### Question 11)

فایل asm. حاوی کد های اسمبلی نوشته شده توسط برنامه نویس است و برنامه نویس این فایل را ایجاد میکند در حالی که سایر فایل های hex , .eep , .obj , .lst. به صورت خودکار توسط اسمبلر ایجاد میگردد از میان این فایل ها فایل hex. حاوی اطلاعات FLASH و فایل eep. حاوی اطلاعات EEPROM می باشد.

به علاوه فایل lst. حاوی اطلاعاتی پیرامون پروسه ی اسمبلی است و بخش هایی از جمله :

warnings and errors , page header , command line , source listing , symbol table , cross reference را داراست.

فایل های obj. شامل کدهای کامپایل شده ای هستند که در آن ها به هیچ کدام از متغیرها یا بلوک های کد، آدرسی در حافظه نسبت داده نشده است. در این جا نیاز به ابزار دیگری به نام linker است . وظیفه ی لینکر این است که فایل های object را باهم دیگر ترکیب کرده و فایل نهایی قابل اجرا با پسوند hex ایجاد کند. linker اطلاعات فایل obj. و اطلاعات نقشه حافظه را از یک فایل سند لینکر برداشته و به متغیرها آدرس نسبت می دهد و بلوک های کد را به بهترین شکل مرتب کرده تا با حافظه منطبق شود. سند لینکر تمام آدرس های ثابت رجیسترهای سخت افزار و حافظه برنامه را دارد.

پروسه اجرای کد مطابق زیر است:



# OPCODE

## Question 12)

با توجه به کد های هگز که برای دستورات مختلف وجود دارد نتیجه تبدیل این کد ها به اعداد هگز به صورت زیر است:

MOV R24,R7 : 2D87

MOV:

$Rd \leftarrow Rr$

**16 bit Opcode:**

0010 11rd dddd rrrr

CBR R24,0b00001111 : 7F80

کد این دستور به صورت 708F بوده ولی این دستور بایت اول و سوم را به صورت مکمل یک می نویسد:

$1111\ 1111 - 0000\ 1111 = 1111\ 0000$

SWAP R24 : 9582

**Operation:**

(i)  $R(7-4) \leftarrow R(3-0)$ ,  $R(3-0) \leftarrow R(7-4)$

**16 bit Opcode:**

1001 010d dddd 0010

rcall BCD\_to\_7SEG : D004

**Operation:**

$PC \leftarrow PC + k + 1$

**16 bit Opcode:**

1101 kkkk kkkk kkkk

CBI PORTA, 7 : 98DF

**Operation:**

$I/O(P,b) \leftarrow 0$

**16 bit Opcode:**

1001 1000 pppp pbbb

از آنجایی که آدرس IO/reg به نام portA برابر \$1B است پس کد این دستور به صورت  
1001 1000 1101 1111 در می آید.

CALL WriteDisplay : 940E 000F

**Operation:**

$PC \leftarrow K$

**32 bit Opcode:**

1001 010k kkkk 111k

kkkk kkkk kkkk kkkk

آدرس writeDisplay را در انتهای برنامه در نظر میگیریم. (CALL دستور ۴ بایتی است)

LDI R31, 200 (0xC8) : ECF8

**Operation:**

$Rd \leftarrow K$

**16 bit Opcode:**

1110 KKKK dddd KKKK

LDI ZL, LOW(TABLE<<1) : E1EE

LDI ZH, HIGH(TABLE<<1) : E0F0

فرض کرده ایم برچسب TABLE به کلمه شماره 0X0F اشاره می کند. مقادیر رجیسترهای ZH و ZL  
به ترتیب معادل R31 و R30 هستند.

CLR R1 : 2411

**Operation:**

$Rd \leftarrow Rd \oplus Rd$

**16 bit Opcode:** (see EOR Rd,Rd)

0010 01dd dddd dddd

دستور CLR معادل R1 XOR R1 است پس در دو بایت آخر، آدرس R1 تکرار می شود.

ADD ZL,R24 : 0FE8

**Operation:**

$Rd \leftarrow Rd + Rr$

**16 bit Opcode:**

0000 11rd dddd rrrr

ADDC ZH,R1 : 1DF1

**Operation:**

$Rd \leftarrow Rd + Rr + C$

**16 bit Opcode:**

0001 11rd dddd rrrr

LPM R24,Z : 9184

**Operation:**

$Rd \leftarrow Rd + Z$

**16 bit Opcode:**

1001 000d dddd 0100

RET : 9508

$PC(15-0) \leftarrow STACK$

$PC(21-0) \leftarrow STACK$

**16 bit Opcode:**

1001 0101 0XX0 1000



با توجه به فرضیات ذکر شده قطعه کد فوق را به صورت زیر تکمیل کرده ایم :

```
MOV R24,R7
CBR R24,0B00001111
SWAP R24
RCALL BCD_TO_7SEG
CBI PORTA,7
CALL WRITEDISPLAY
LDI R31,200
BCD_TO_7SEG:
LDI ZL,LOW(TABLE<<1)
LDI ZH,HIGH(TABLE<<1)
CLR R1
ADD ZL, R24
ADC ZH, R1
LPM R24, Z
RET
WRITEDISPLAY:
.ORG 0X0F
TABLE:
```

### Question 13)

ابتدا کد ها را از حالت little endian خارج می کنیم و سپس دستورات متناظر آن ها را می یابیم.

```
.ORG 0x0020
E249      →LDI R20,41
2F54      →MOV R21,R20
705F      →ANDI R21,15
6350      →ORI R21,48
2F64      →MOV R22,R20
9562      →SWAP R22
706F      →R22,0x0F
6360      →R22,0x30
940C 0028 →LABEL: JMP LABEL
```

### Question 14)

0x25 -> 37 decimal

$37 * 2 = 74$

74 decimal -> 0x4A

آدرس محل شروع کلمه 0x25 است که معادل بایت هفتاد و چهارم (0x4A) است.

داده های بعد از دستور DB باید به صورت تعداد زوجی از بایت ها باشند. اگر تعداد بایت ها فرد باشد اسمبلر بایت آخر را با صفر پر می کند. از آنجایی که بعد از اولین DB به تعداد ۳ بایت داده داریم، اسمبلر یک بایت ۰۰ بعد از آنها اضافه می کند. بعد از دومین DB هم یک بایت داده داریم پس این اتفاق تکرار می شود. برای نوشتن رشته HERE در حافظه، کد های اسکی کاراکتر ها به ترتیب نوشته می شوند. دستور آخر DW است که تک تک داده ها را به صورت دو بایتی می نویسد و چون 0x45 یک بایتی است، بعد از آن 00 میگذارد، این دستور کاراکتر های دو بایتی را به صورت little endian می نویسد. بدین ترتیب محتوای حافظه flash از بایت 0x004A به بعد به این صورت است:

03 61 23 00 ff 00 48 45 52 45 14 23 45 00

### Question 15)

خطا ها:

- ۱- رجیستر های SPH و SPL جزو رجیستر های IO هستند که نمی توان مستقیماً عدد داخل آن ها ریخت لذا می بایست ابتدا مقدار HIGH(151) , LOW(151) در داخل یکی از رجیستر های R ریخته شده و سپس مقدار رجیستر R با دستور OUT داخل SPL, SPH قرار گیرند.
- ۲- خط دوم باید به صورت **LDI SPH,HIGH(151)** باشد که به اشتباه SPL نوشته شده.
- ۳- رجیسترهای R0 و YH, ZH مقدار دهی نشده، ما با فرض صفر کردن همه مسئله را حل می کنیم.

۴- داخل تابع DELAY ابتدا آدرس برگشت داخل استک push می شود. سپس طبق دستورات مقادیر R12 و R22 در استک push می شوند اما این دو مقدار pop نمی شوند پس هنگام return مقادیر این دو رجیستر به عنوان آدرس برگشت pop می شوند که چون مقدار حاصل برابر 0x55 می شود و از محل کد و محل داده ها بسیار دور است. پس برای برگشت به ادامه ی کد باید قبل از RET، این دو خط کد را نوشت.

POP R20  
POP R21

۵- به جای دستور LDD R2,Z+ می بایست از LD R2,Z+ و به جای STD Y+,R2 می بایست از ST Y+,R2 استفاده کنیم.

در صورتی که اشکالات کد برطرف شود، این کد می خواهد مقادیر موجود در حافظه RAM از آدرس 0X66+K تا ۵ خانه بعد را ابتدا یک به یک به اندازه یک بیت شیفت گردشی به راست داده و سپس در خانه هایی از RAM با شروع از آدرس 0X7F+K تا ۵ خانه بعد از آن ذخیره نماید. K همان عدد ورودی است که از طریق PINB دریافت می شود. با فرض آن که مقدار K برابر صفر باشد آنگاه محتویات RAM از آدرس 0X66 تا آدرس 0X6A تک به تک پس از یک بیت شیفت گردشی به راست در آدرس 0X7F تا 0X83 حافظه RAM ذخیره می شوند.  
در این صورت محتویات RAM از خانه 0X7F تا 0X83 مطابق زیر خواهد بود:

C = 0

'1' = 110001 → 011000 → C = 1 → 0x18 → 0X7F RAM

'2' = 110010 → 111001 → C = 0 → 0x39 → 0X80 RAM

'3' = 110011 → 011001 → C = 1 → 0x19 → 0X81 RAM

'4' = 110100 → 111010 → C = 0 → 0x3A → 0X82 RAM

'5' = 110101 → 011010 → C = 1 → 0x1A → 0X83 RAM

قطعه کد فوق را به صورت زیر تصحیح کرده ایم :

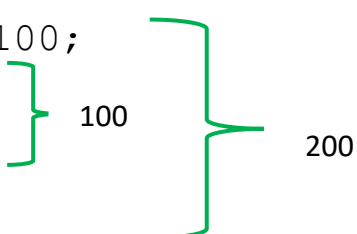
```
.ORG 0x23
LDI R16, LOW(151)
OUT SPL,R16
LDI R16, HIGH(151)
OUT SPH,R16
LDI R21,5
MOV R20,R21
LDI R22,0x00
MOV R0,R22
LDI R31,0X00
LDI R29,0X00
OUT DDRB,R22
CALL DELAY
IN R3,PINB
LDI R30,102
LDI YL,127
ADD R30,R3
ADC R31,R0
ADD R28,R3
ADC R29,R0
HERE: LD R2,Z+
ROR R2
ST Y+,R2
DEC R20
BRNE HERE
OVER: JMP OVER
```

```
DELAY:
NOP
NOP
PUSH R21
PUSH R20
NOP
POP R20
POP R21
RET
```

## PROGRAMMING I

### Question 16)

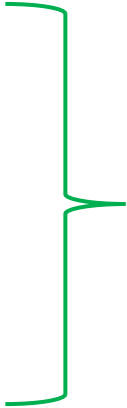
```
LDI R20,200;
BACK: LDI R21,100;
HERE: DEC R21;
BRNE HERE;
DEC R20;
BRNE BACK;
```




the code repeats  $200 \times 100 = 20000$  times.

### Question 17)

DELAY: LDI R20, 200

BACK: LDI R25, 100	→ one clock		200
NOP	→ one clock		
NOP	→ one clock		
NOP	→ one clock		
HERE: DEC R25	→ one clock		
BRNE HERE	→ one or two clock		
DEC R20	→ one clock		
BRNE BACK	→ one or two clock		
RET	→ four clock		

 100

Frequency : 20MHZ → each cycle takes 0.05 μsec

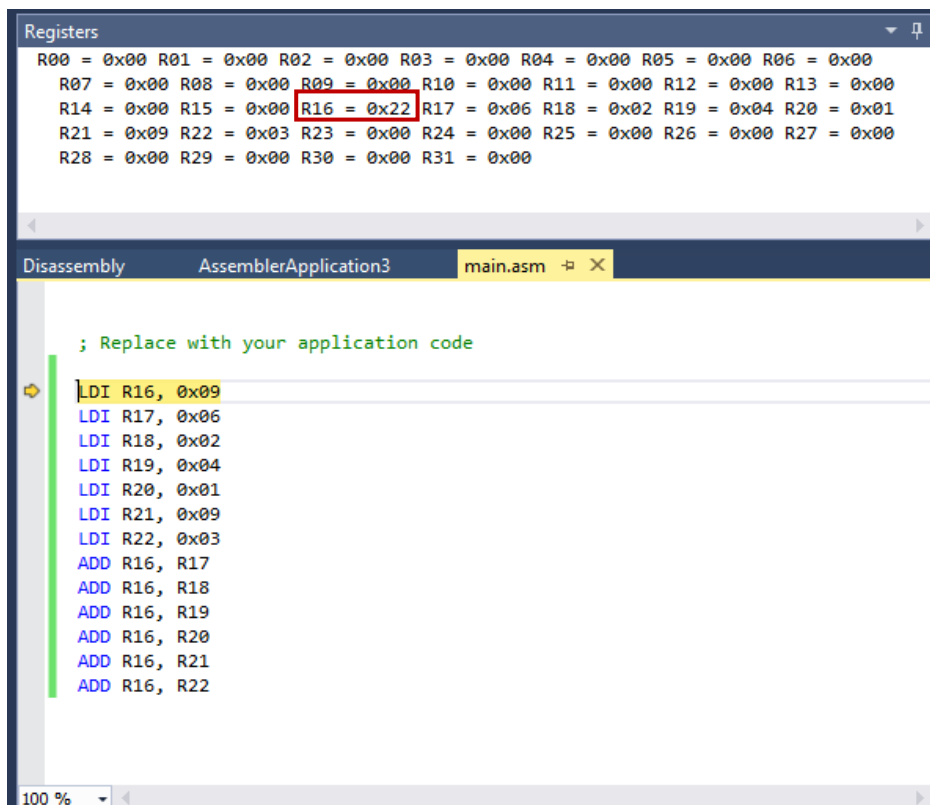
$$1 + ((100 + 99 * 2 + 1) * 200) + (200 + 199 * 2 + 1) + (3 * 200) = 61000$$

زمان واقعی تاخیری که subroutine ایجاد می کند مطابق زیر خواهد بود :

$$61000 * 0.05 = 3050 \mu\text{sec}$$

### Question 18)

```
LDI R16, 0x09
LDI R17, 0x06
LDI R18, 0x02
LDI R19, 0x04
LDI R20, 0x01
LDI R21, 0x09
LDI R22, 0x03
ADD R16, R17
ADD R16, R18
ADD R16, R19
ADD R16, R20
ADD R16, R21
ADD R16, R22
```



### Question 19)

```
LDI R20, 0
LDI R21, 1
LDI R22, 2
LDI R23, 3
SUB R20, R18
BRNE TESTSUB
ADD R16, R17
MOV R0, R16
JMP END
TESTSUB:
SUB R21, R18
BRNE TESTMUL
SUB R16, R17
MOV R0, R16
JMP END
TESTMUL:
SUB R22, R18
BRNE TESTDIV
```

```
MUL R16, R17
JMP END
TESTDIV:
SUB R23, R18
BRNE END
LDI R24, 0X00
START:
SUB R16, R17
BRCC CONTINUE
ADD R16, R17
MOV R0, R24
MOV R1, R16
JMP END
CONTINUE:
INC R24
JMP START
END: RJMP END
```

Registers

R00 = 0x09 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00  
R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00  
R14 = 0x00 R15 = 0x00 R16 = 0x09 R17 = 0x03 R18 = 0x00 R19 = 0x00 R20 = 0x00  
R21 = 0x01 R22 = 0x02 R23 = 0x03 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00  
R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x00

Disassembly AssemblerApplication3 main.asm

LDI R16, 6  
LDI R17, 3  
LDI R18, 0  
LDI R20, 0  
LDI R21, 1  
LDI R22, 2  
LDI R23, 3  
SUB R20, R18  
BRNE TESTSUB  
ADD R16, R17  
MOV R0, R16  
JMP END  
TESTSUB:  
SUB R21, R18  
BRNE TESTMUL  
SUB R16, R17  
MOV R0, R16  
JMP END  
TESTMUL:

100 %

Registers

R00 = 0x03 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00  
R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00  
R14 = 0x00 R15 = 0x00 R16 = 0x03 R17 = 0x03 R18 = 0x01 R19 = 0x00 R20 = 0xFF  
R21 = 0x00 R22 = 0x02 R23 = 0x03 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00  
R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x00

Disassembly AssemblerApplication3 main.asm

LDI R16, 6  
LDI R17, 3  
LDI R18, 1  
LDI R20, 0  
LDI R21, 1  
LDI R22, 2  
LDI R23, 3  
SUB R20, R18  
BRNE TESTSUB  
ADD R16, R17  
MOV R0, R16  
JMP END  
TESTSUB:  
SUB R21, R18  
BRNE TESTMUL  
SUB R16, R17  
MOV R0, R16  
JMP END  
TESTMUL:

100 %

Registers

R00 = 0x12 R01 = 0x00 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00  
R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00  
R14 = 0x00 R15 = 0x00 R16 = 0x06 R17 = 0x03 R18 = 0x02 R19 = 0x00 R20 = 0xFE  
R21 = 0xFF R22 = 0x00 R23 = 0x03 R24 = 0x00 R25 = 0x00 R26 = 0x00 R27 = 0x00  
R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x00

Disassembly AssemblerApplication3 main.asm

```
LDI R16, 6
LDI R17, 3
LDI R18, 2
LDI R20, 0
LDI R21, 1
LDI R22, 2
LDI R23, 3
SUB R20, R18
BRNE TESTSUB
ADD R16, R17
MOV R0, R16
JMP END
TESTSUB:
SUB R21, R18
BRNE TESTMUL
SUB R16, R17
MOV R0, R16
JMP END
TESTMUL:
```

Registers

R00 = 0x01 R01 = 0x01 R02 = 0x00 R03 = 0x00 R04 = 0x00 R05 = 0x00 R06 = 0x00  
R07 = 0x00 R08 = 0x00 R09 = 0x00 R10 = 0x00 R11 = 0x00 R12 = 0x00 R13 = 0x00  
R14 = 0x00 R15 = 0x00 R16 = 0x01 R17 = 0x05 R18 = 0x03 R19 = 0x00 R20 = 0xFD  
R21 = 0xFE R22 = 0xFF R23 = 0x00 R24 = 0x01 R25 = 0x00 R26 = 0x00 R27 = 0x00  
R28 = 0x00 R29 = 0x00 R30 = 0x00 R31 = 0x00

Disassembly AssemblerApplication3 main.asm

```
LDI R16, 6
LDI R17, 5
LDI R18, 3
LDI R20, 0
LDI R21, 1
LDI R22, 2
LDI R23, 3
SUB R20, R18
BRNE TESTSUB
ADD R16, R17
MOV R0, R16
JMP END
TESTSUB:
SUB R21, R18
BRNE TESTMUL
SUB R16, R17
MOV R0, R16
JMP END
TESTMUL:
```

R0 → result  
R1 → remainder