

در ابتدا باید بعضی از رجیستر ها را مقدار دهی اولیه کنیم

مثلا برای برای دریافت عدد از ورودی ابتدا `ddrA` را صفر میکنیم و برای فعال سازی `pull-up` مقدار ۲۵۵ را بروی `portA` قرار میدهیم

که کد آن به شرح زیر است :

```
ldi r16 , 0
out ddrA , r16
ldi r16 , 0xff
out portA , r16
```

سپس مقدار ۱۰ را به رجیستر ۱۷ می دهیم تا نقش شمارنده را داشته باشد

```
ldi r17 , 10
```

و همچنین ادرس خانه هایی از حافظه که می خواهیم عدد ها را به تناسب زوج و فرد بودن در آن ها ذخیره کنیم در رجیستر های `X` و `Y` میگذاریم

```
ldi x1 , low(0x060)
ldi xh , high(0x060)
ldi y1 , low(0x070)
ldi yh , high(0x070)
```

در بدنه اصلی ابتدا عدد را از ورودی دریافت میکنیم

```
in r16 , pina
```

سپس یک مقدار `backup` از عدد می گیریم تا در صورتی که در پردازش عدد ، عدد اصلی دچار تغییر شد بتوانیم مقدار درست را در `RAM` ذخیره کنیم .

```
mov r18 , r16
```

می دانیم که عدد زوج باینری در کم ارزش ترین بیت خود (`LSB`) مقدار صفر را دارد و عدد فرد مقدار ۱ را دارد .

پس برای تصمیم گیری که عدد ورودی زوج یا فرد است نیاز به بررسی `LSB` داریم .

عدد ورودی را با مقدار باینری ۱، AND، میکنیم

با این کار اگر عدد ورودی در LSB خود ۱ داشته باشد

با AND کردن با مقدار ۱، مقدار ۱ را تولید می کند و در صورتی که در LSB، صفر باشد با AND کردن با مقدار ۱، صفر را تولید میکند

```
andi r16 , 0x00000001
```

با دستور BREQ بروی مقدار حاصل از عملیات AND، تصمیم گیری میکنیم

```
breq EVEN
```

پس از آن با توجه به اینکه عددی ورودی زوج است یا فرد با استفاده از دستور `st` مقداری اصلی را که بروی یک رجیستر دیگر کپی کردیم بر میداریم و در خانه ای از حافظه که پوینتر به آن اشاره می کند می ریزیم و مقدار پوینتر را یک واحد می افزایشیم

```
st y+ , r18
```

اگر عدد فرد باشد پس از دستور `breq` به خط بعدی می رود

دستور `breq` در واقع می گوید که اگر عدد مساوی با صفر است به `EVEN` برو و دستورات را از آنجا انجام بده

حال در صورت فرد بودن حاصل `ANDI` مساوی با صفر نیست و دستور `breq` را اجرا نمی کند و به خط بعدی یعنی:

```
st y+ , r18
```

می رود که پس از ذخیره مقدار موجود در رجیستر ۱۸ به `loop` می پریم که در آنجا یک مقدار از شمارنده کم کنیم و دوباره از نو عددی از ورودی بگیریم.

اما در صورتی که عدد ورودی زوج باشد پس از `ANDI` مقدار آن صفر میشود و با دستور `breq` به `EVEN` میرویم و با استفاده از دستور `st` مقدار موجود در رجیستر ۱۸ که همان عدد اصلی ورودی است را به ادرسی که پوینتر به آن اشاره میکند میریزیم و مقدار پوینتر را یک واحد می افزایشیم

`EVEN :`

```
st x+ , r18
```

در خط بعدی آن به loop میرویم و از شمارنده یک مقدار کسر میکنیم

سپس در پایان یک عدد از شمارنده کم میکنیم و در هر حلقه هم چک می کنیم که آیا مقدار شمارنده به صفر رسیده یا نه در صورتی که به صفر رسیده باشد از برنامه خارج می شویم

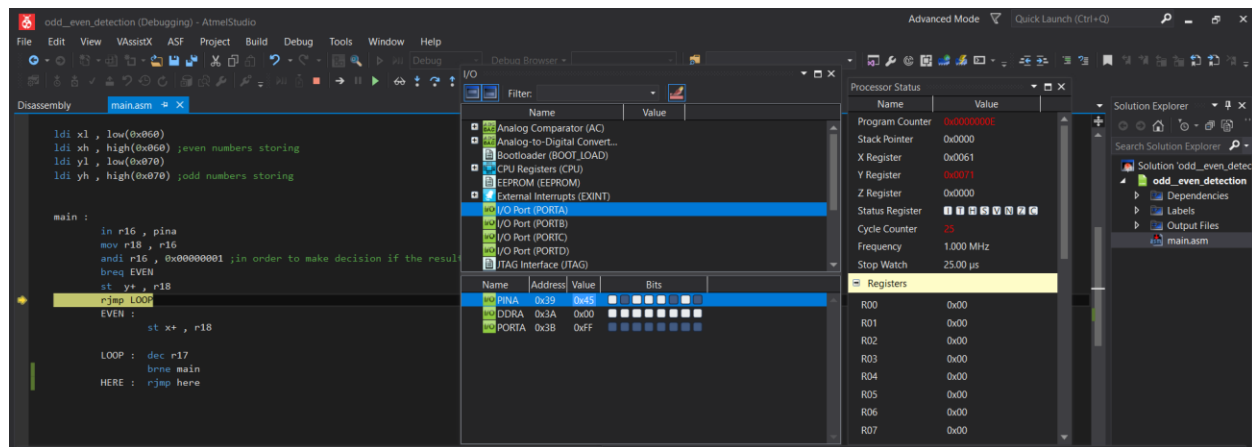
```
LOOP :      dec r17
           brne main
```

که خارج شدن از با نوشتن یک حلقه بی نهایت پیاده سازی میکنیم که میکرو کنترلر بی تکلیف نباشد

```
HERE :      rjmp here
```

• نمونه ای از مراحل اجرا :

در قسمت debug برنامه مقادیری را به صورت دستی وارد می کنیم تا از عملکرد صحیح کد اطمینان حاصل کنیم :

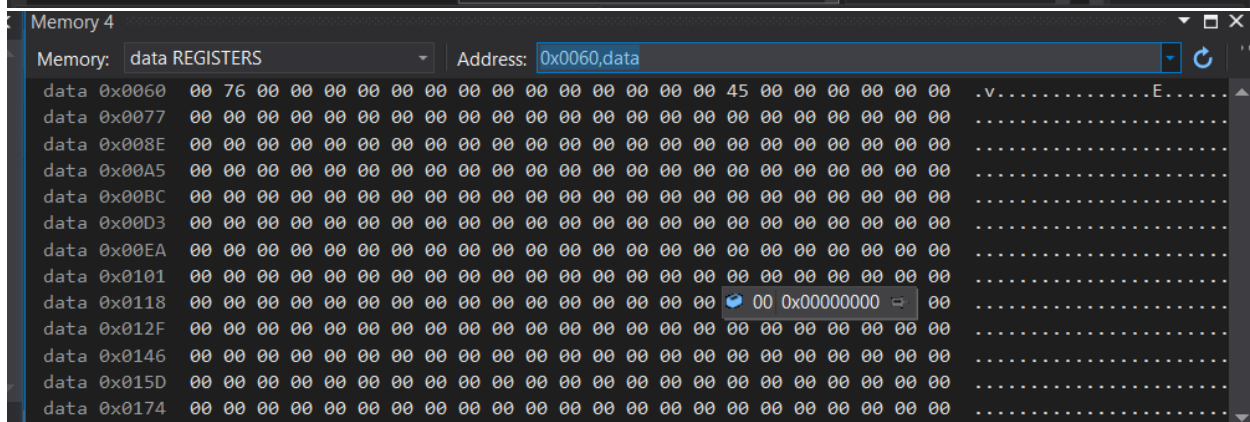
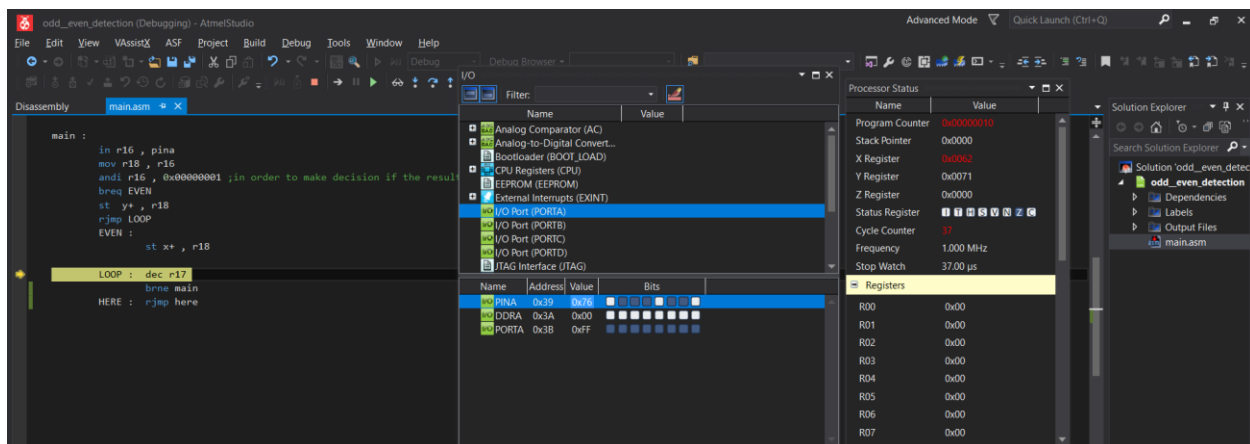


مقدار 0x45 را در pinA وارد کردیم




همان طور که می بینید در نتیجه در خانه 0x70 مقدار ۴۵ ریخته شده است.

یا مثلا مقدار 0x76 که مقدار زوج می باشد :



مقدار داده شده به رجیستر ها :

Watch 1			▼ □ ✕
Name	Value	Type	▲
 r16	0x00	byte(regi	
 r17	0x08	byte(regi	
 r18	0x76	byte(regi	