PERANCANGAN KEYPAD MESIN FOTOCOPY SEDERHANA MENGGUNAKAN ATMEL AVR ATmega8535

Oleh :
AL FARISI

(alfarisi@linuxmail.org **)**

http://www.alfarisi.web.ugm.ac.id
http://www.alfarisi.co.nr
http://www.alfarisi.tk

PENDAHULUAN

Pada kesempatan ini penulis mencoba untuk merancang dan memodifikasi keypad di sebuah mesin fotocopy sehingga dapat bekerja dengan spesifikasi berikut :

1. Keypad berupa push switch, 3 buah di antaranya masing-masing jika ditekan memberikan input 10 lembar, 25 lembar dan 50 lembar fotokopi.
2. Sebuah push switch lagi dapat digunakan untuk menginterupsi proses sehingga mesin berhenti mengkopi.
3. Sebuah push switch yang lain untuk melanjutkan kerja mesin setelah diinterupsi.
4. Untuk menunjukkan posisi jumlah kopian ada led yang jumlahnya 5 yang menyala sesuai jumlah cacahan kelipatan 10.

Mikrokontroller yang digunakan adalah AVR ATmega8535. Penulis berasumsi, pembaca sudah cukup mengenal mikrokontroller ATmega8535 sehingga tidak akan dibahas lebih terperinci lagi. Di bawah ini penulis sajikan tabel konfigurasi input/output dan tabel vektor interupsi pada ATmega8535 yang mungkin berguna sebagai referensi.

Konfigurasi Input/Output

	DDR bit = 1	**DDR bit = 0**
Port bit = 1	Output High	Input Pull-Up
Port bit = 0	Output Low	Input Floating

Tabel Vektor Interupsi

Vector No.	**Program Address**	**Source**	**Interrupt Definition**
1	0x000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete

16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	INT2	External Interrupt Request 2
20	0x013	TIMER0 COMP	Timer/Counter0 Compare Match
21	0x014	SPM_RDY	Store Program Memory Ready

DESKRIPSI RANCANGAN

Berikut ini konfigurasi pin/port ATmega8535 yang dibuat oleh penulis.

- Untuk memberikan delay antar setiap lembar fotocopy dilakukan dengan menggunakan Timer/Counter1 16-bit.
- Pin D bit 4 untuk memberikan input 10 lembar fotocopy.
- Pin D bit 5 untuk memberikan input 25 lembar fotocopy.
- Pin D bit 6 untuk memberikan input 50 lembar fotocopy (jumlah maksimum).
- Pin D bit 2 merupakan interupsi eksternal 0, yang apabila ditekan maka akan menghentikan proses fotocopy.
- Pind D bit 1 digunakan untuk melanjutkan proses fotocopy setelah mesin diinterupsi.
- Port C bit 0 digunakan untuk memberikan sinyal ke mesin fotocopy agar melakukan proses fotocopy.
- Port C bit 7 untuk menyalakan LED pertama saat jumlah fotocopy mencapai 10 lembar.
- Port C bit 6 untuk menyalakan LED kedua saat jumlah fotocopy mencapai 20 lembar.
- Port C bit 5 untuk menyalakan LED ketiga saat jumlah fotocopy mencapai 30 lembar.
- Port C bit 4 untuk menyalakan LED keempat saat jumlah fotocopy mencapai 40 lembar.
- Port C bit 3 untuk menyalakan LED kelima saat jumlah fotocopy mencapai 50 lembar (jumlah maksimum).

ALAT DAN BAHAN

1. Komputer dan *peripheral*-nya yang berjalan di atas Sistem Operasi Windows XP Proffesional Service Pack 2.
2. AVR Studio 4.11 beserta AVR Simulator.
3. Berbagai referensi yang didapat dari website, datasheet maupun sumber-sumber lainnya.

PROGRAM ASSEMBLY

Untuk merealisasikan rancangan tersebut di atas, dibuat program dalam bahasa assembly sebagai berikut.

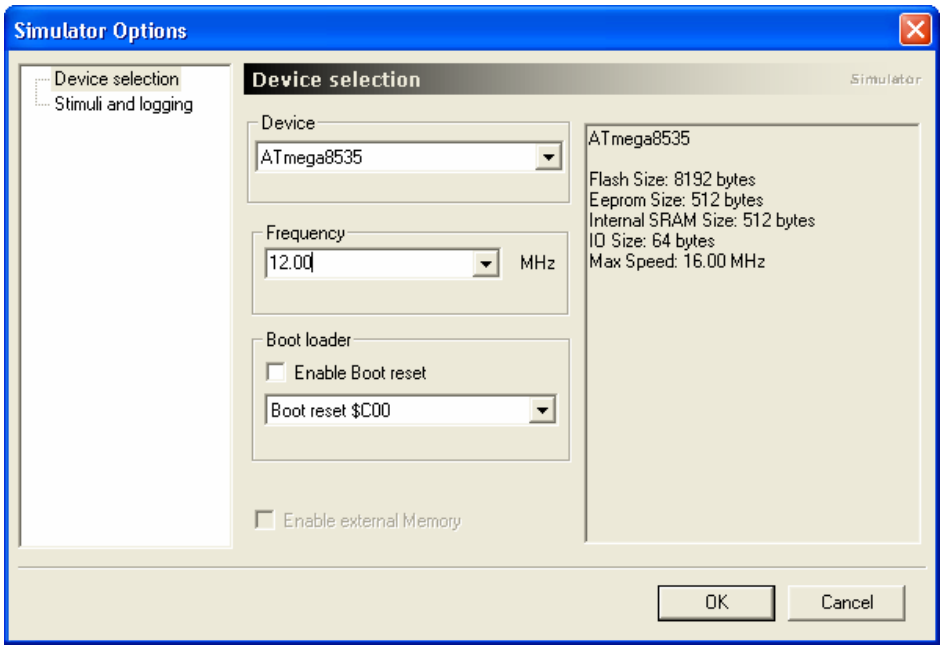
```
;---------------------------------------------- begin ----------------------------------------------
.def jumlahSaiki = r16
.def jumlahCopy = r17
.def data = r18                 ;register sementara, penyimpan data
.equ waktu = 0xf48e             ;untuk delay = 0,25 s dgn f=12 MHz

.include "m8535def.inc"
.org 0x0000
rjmp main                       ;alamat 0x0000 reset
rjmp stop                       ;alamat 0x0001 external interupt 0

main:
ldi data,low(ramend)
out spl,data
ldi data,high(ramend)
out sph,data

ldi data,0b01000000             ;enable external interupt 0
out GICR,data
ldi data,0x00                   ;memberi logic 0 pada ISC01 dan ISC00
out MCUCR,data
sei                             ;enable global interupt

ldi data,0x00
out ddrd,data                   ;pind sebagai input
ldi data,0xff
out ddrc,data                   ;portc sebagai output
out portd,data

awal:
ldi data,0x00
mov jumlahSaiki,data            ;nilai awal jumlahSaiki = 0
ldi data,0xff
out portc,data                  ;LED mati dan tidak ada signal ke mesin

cek10:
sbic pind,4                     ;cek pind 4 ditekan atau tidak
rjmp cek25                      ;jika tidak, lompat ke subrutin cek25
ldi jumlahCopy,0x0a             ;jika ya, jumlahCopy = 10
rjmp copy                       ;lompat ke subrutin copy

cek25:
sbic pind,5                     ;cek pind 5 ditekan atau tidak
rjmp cek50                      ;jika tidak, lompat ke subrutin cek50
ldi jumlahCopy,0x19             ;jika ya, jumlahCopy = 25
rjmp copy                       ;lompat ke subrutin copy

cek50:
sbic pind,6                     ;cek pind 6 ditekan atau tidak
rjmp cek10                      ;jika tidak, kembali ke subrutin cek10
```

```
ldi jumlahCopy,0x32                ;jika ya, jumlahCopy = 50

copy:
inc jumlahSaiki
cbi portc,0                        ;signal utk meng-copy (logic 0)
sbi portc,0                        ;matikan kembali signal (logic 1)
rcall delay                        ;delay sebentar, di simulasi cukup lama
rcall led1                         ;panggil subrutin led
cp jumlahSaiki,jumlahCopy
brne copy
rjmp awal

;---------------------------------------------------------------------
; subrutin untuk ngecek jumlah lembar dan nyalain LED
;---------------------------------------------------------------------

led1:
cpi jumlahSaiki,0x0a               ;bandingkan jumlahSaiki dengan 10
brne led2                          ;jika tidak bernilai 10 -> ke subrutin led2
cbi portc,7                        ;nyalakan led pada portc 7

led2:
cpi jumlahSaiki,0x14               ;bandingkan jumlahSaiki dengan 20
brne led3                          ;jika tidak bernilai 20 -> ke subrutin led3
cbi portc,6                        ;nyalakan led pada portc 6

led3:
cpi jumlahSaiki,0x1e               ;bandingkan jumlahSaiki dengan 30
brne led4                          ;jika tidak bernilai 30 -> ke subrutin led4
cbi portc,5                        ;nyalakan led pada portc 5

led4:
cpi jumlahSaiki,0x28               ;bandingkan jumlahSaiki dengan 40
brne led5                          ;jika tidak bernilai 40 -> ke subrutin led5
cbi portc,4                        ;nyalakan led pada portc 4

led5:
ldi data,0x32
cpse jumlahSaiki,data              ;bandingkan jumlahSaiki dengan 50
ret                                ;kembali tanpa menyalakan LED
cbi portc,3                        ;nyalakan led pada portc 3
ret

;---------------------------------------------------------------------
; subrutin delay 0,25 detik = 250.000 us = 3.000.000 cycle
;---------------------------------------------------------------------

delay:
ldi data,0x00                      ;disable overflow interupt Timer1
out TIMSK,data
ldi data,high(waktu)               ;isi TCNT1 dengan 0xF48E
out TCNT1H,data
ldi data,low(waktu)
out TCNT1L,data
ldi data,0b00000101                ;jalankan Timer1 dengan prescaler : 1024
out TCCR1B,data
```

```
looptime:
in data,TIFR                    ;tunggu sampai Timer1 overflow flag set
sbrs data,TOV1
rjmp looptime
ldi data,0b00000100             ;overflow flag dinolkan
out TIFR,data
ldi data,0x00                   ;stop Timer1
out TCCR1B,data
ret

stop:                           ;subrutin eksternal interupt 0
sbic pind,1                     ;tunggu sampai pind 1 ditekan
rjmp stop
reti
;---------------------------------------------- end ----------------------------------------------
```

PEMBAHASAN PROGRAM

Sebelumnya, perlu diketahui bahwa pada simulasi dengan menggunakan AVR Studio, penulis menggunakan frekuensi 12 MHz. Pengesetan frekuensi dapat dilakukan saat program berada dalam mode **Debug**. Saat sedang proses *debugging*, pilih *Debug → AVR Simulator Options*, kemudian pada jendela yang muncul dapat dipilih atau ditulis frekuensi yang kita inginkan.



Pada bagian awal program dilakukan pendefinisian terhadap register, nilai data dan jenis mikrokontroller yang dipakai, yaitu :

```
.def jumlahSaiki = r16
.def jumlahCopy = r17
.def data = r18                 ;register sementara, penyimpan data
```

```
.equ waktu = 0xf48e                    ;untuk delay = 0,25 s dgn f=12 MHz
.include "m8535def.inc"
```

Pendefinisian pada baris 1 sampai 4 di atas tidak harus dilakukan. Hal tersebut hanya untuk mempermudah pengaksesan register dan data. Register r16 didefinisikan kembali sebagai *jumlahSaiki*, yaitu register yang berisi nilai jumlah lembar yang sudah difotocopy. Register r17 diberi nama *jumlahCopy*, berfungsi sebagai register yang berisi jumlah keseluruhan lembar yang ingin difotocopy. Sedangkan sebagai register untuk mengakses data digunakan r18 yang diberi nama *data*. Baris kelima berfungsi untuk meng-*include*-kan file *m8535def.inc*. Bila kita menggunakan ATmega8535, maka file ini harus diikutsertakan. Dengan meng-*include*-kan file ini maka semua nama register I/O beserta bit-bitnya dapat digunakan.

```
.org 0x0000
rjmp main                              ;alamat 0x0000 reset
rjmp stop                              ;alamat 0x0001 external interupt 0
```

.org 0x0000 digunakan untuk memberitahu bahwa instruksi selanjutnya akan ditempatkan pada alamat 0000. Jadi, pada alamat 0000 akan ditempati oleh instruksi *rjmp main*. Instruksi ini memberi perintah untuk meloncat ke subrutin *main*. Alamat ini juga merupakan alamat vektor interupsi pertama, yaitu RESET. Jadi begitu ada interupsi reset maka program akan meloncat ke alamat ini dan menjalankan kembali subrutin *main*. Kemudian, pada alamat 0001 ditempati oleh instruksi *rjmp stop*. Instruksi ini akan memberi perintah untuk meloncat ke subrutin *stop*. Alamat 0001 adalah alamat vektor interupsi kedua, yaitu interupsi eksternal 0 (melalui pin INT0 / pin D bit 2). Jadi perintah ini hanya akan dieksekusi bila pin INT0 ditekan.

Program pada subrutin *main* adalah sebagai berikut.

```
main:
ldi data,low(ramend)
out spl,data
ldi data,high(ramend)
out sph,data

ldi data,0b01000000            ;enable external interupt 0
out GICR,data
ldi data,0x00                  ;memberi logic 0 pada ISC01 dan ISC00
out MCUCR,data
sei                            ;enable global interupt

ldi data,0x00
out ddrd,data                  ;pind sebagai input
ldi data,0xff
out ddrc,data                  ;portc sebagai output
out portd,data
```

Baris kedua sampai keempat pada cuplikan program di atas digunakan untuk mengisi *Stack Pointer* (SPL dan SPH) dengan alamat terakhir SRAM. Baris selanjutnya digunakan untuk meng-*enable* interupsi eksternal 0, yaitu dengan cara

memberi logic 1 pada bit 6 (INT0) register GICR (*General Interrupt Control Register*).

INT1	INT0	INT2	4	3	2	IVSEL	IVCE

GICR

Keterangan :

- INT1 (External Interrupt Request 1 Enable)
- INT0 (External Interrupt Request 0 Enable)
- INT2 (External Interrupt Request 2 Enable)

Konfigurasi terhadap interupsi ekternal 0 juga perlu dilakukan, untuk memberitahu program dalam keadaan seperti apa yang akan menyebabkan terjadinya interupsi. Hal ini dapat dilakukan melalui register MCUCR.

SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00

MCUCR

Berikut ini konfigurasi bit ISC01 dan ISC00 untuk interupsi eksternal 0 yang penulis peroleh dari *Datasheet ATmega8535*.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Dengan memberi logic 0 pada bit ISC01 dan ISC00 maka interupsi eksternal 0 hanya terjadi bila pin INT berlogic *low* (dalam keadaan ditekan). Namun, interupsi tetap tidak bisa dilakukan jika *Global Interupts* di-disable. Untuk meng-enable global interupt dilakukan dengan menge-set bit ketujuh (I) pada register SREG melalui instruksi *sei*.

Pada baris program selanjutnya dilakukan konfigurasi pin / port mana yang dijadikan sebagai input maupun output. Yang berfungsi sebagai input adalah pin D. Konfigurasi dilakukan dengan memberikan logic 0 pada DDR dan logic 1 pada port D. Sedangkan sebagai output adalah port C, yaitu dengan memberikan logic 1 pada DDR. Selanjutnya, program akan masuk pada subrutin *awal*.

```
awal:
ldi data,0x00
mov jumlahSaiki,data          ;nilai awal jumlahSaiki = 0
ldi data,0xff
out portc,data                ;LED mati dan tidak ada signal ke mesin
```

Pada program di atas kita mengisi nilai awal *jumlahSaiki* (register 16) dengan 0. Penulis juga menge-set LED dalam keadaan mati dan tidak ada signal, yaitu dengan memberikan logic 1 pada semua bit port C. Setelah itu program akan mengecek apakah ada input dari user atau tidak.

```
cek10:
sbic pind,4                 ;cek pind 4 ditekan atau tidak
rjmp cek25                  ;jika tidak, lompat ke subrutin cek25
ldi jumlahCopy,0x0a         ;jika ya, jumlahCopy = 10
rjmp copy                   ;lompat ke subrutin copy

cek25:
sbic pind,5                 ;cek pind 5 ditekan atau tidak
rjmp cek50                  ;jika tidak, lompat ke subrutin cek50
ldi jumlahCopy,0x19         ;jika ya, jumlahCopy = 25
rjmp copy                   ;lompat ke subrutin copy

cek50:
sbic pind,6                 ;cek pind 6 ditekan atau tidak
rjmp cek10                  ;jika tidak, kembali ke subrutin cek10
ldi jumlahCopy,0x32         ;jika ya, jumlahCopy = 50
```

Program di atas mengecek keadaan pin D bit 4, 5 dan 6 secara berurutan. Jika bit 4 pin C ditekan (logic 0) maka program akan mengisi *jumlahCopy* (register 17) dengan nilai 10 kemudian meloncat ke subrutin *copy* untuk melakukan proses fotocopy. Tetapi, bila pin C bit 4 tidak ditekan (logic 1) maka program akan melanjutkan pengecekan terhadap bit 5 pin C. Jika bit ini ditekan maka program akan mengisi *jumlahCopy* dengan nilai 25 kemudian meloncat ke subrutin *copy* untuk melakukan proses fotocopy. Dan seandainya tidak ditekan, maka program akan melanjutkan pengecekan terhadap bit 6 pin C. Seperti sebelumnya, jika bit ini ditekan maka program akan mengisi *jumlahCopy* dengan nilai 50 kemudian meloncat ke subrutin *copy* untuk melakukan proses fotocopy. Jika tidak ditekan maka program akan kembali mengulang pengecekan terhadap bit 4 pin C. Proses ini akan terus diulang sampai salah satu bit tersebut ditekan, atau akan berhenti bila terjadi interupsi dari pin INT0 maupun RESET.

```
copy:
inc jumlahSaiki
cbi portc,0                 ;signal utk meng-copy (logic 0)
sbi portc,0                 ;matikan kembali signal (logic 1)
rcall delay                 ;delay sebentar, di simulasi cukup lama
rcall led1                  ;panggil subrutin led
cp jumlahSaiki,jumlahCopy
brne copy
rjmp awal
```

Subrutin di atas merupakan program yang digunakan untuk melakukan proses fotocopy. Program akan meng-*increment* nilai *jumlahSaiki* kemudian memberikan signal ke mesin untuk meng-copy melalui port C bit 0, dengan cara memberi logic *low* melalui perintah *cbi portc, 0*. Lalu, sinyal dimatikan kembali dengan memberikan logic *high* pada bit ini. Program akan memanggil subrutin *delay* dan *led1*. Subrutin *delay* dipakai untuk memberikan jeda antara proses fotocopy tiap lembarnya. Sedangkan subrutin *led1* untuk mengecek jumlah lembar yang telah difotocoy dan menyalakan LED sesuai dengan cacahan kelipatan 10.

Setelah itu, program akan membandingkan nilai *jumlahSaiki* dengan *jumlahCopy*. Jika nilainya tidak sama maka program akan kembali mengulang

subrutin *copy*. Tetapi, jika nilainya telah sama maka program akan menghentikan proses fotocopy dan meloncat ke subrutin *awal* dan kembali menunggu masukan dari user.

Program yang mengatur nyalanya LED adalah sebagai berikut.

```
led1:
cpi jumlahSaiki,0x0a        ;bandingkan jumlahSaiki dengan 10
brne led2                   ;jika tidak bernilai 10 -> ke subrutin led2
cbi portc,7                 ;nyalakan led pada portc 7

led2:
cpi jumlahSaiki,0x14        ;bandingkan jumlahSaiki dengan 20
brne led3                   ;jika tidak bernilai 20 -> ke subrutin led3
cbi portc,6                 ;nyalakan led pada portc 6

led3:
cpi jumlahSaiki,0x1e        ;bandingkan jumlahSaiki dengan 30
brne led4                   ;jika tidak bernilai 30 -> ke subrutin led4
cbi portc,5                 ;nyalakan led pada portc 5

led4:
cpi jumlahSaiki,0x28        ;bandingkan jumlahSaiki dengan 40
brne led5                   ;jika tidak bernilai 40 -> ke subrutin led5
cbi portc,4                 ;nyalakan led pada portc 4

led5:
ldi data,0x32
cpse jumlahSaiki,data       ;bandingkan jumlahSaiki dengan 50
ret                         ;kembali tanpa menyalakan LED
cbi portc,3                 ;nyalakan led pada portc 3
ret
```

Pengecekan juga dilakukan secara urut. Yang pertama, adalah mengecek apakah jumlah lembar yang difotocopy sudah mencapai 10 atau belum. Pengecekan dapat dilakukan dengan cara membandingkan *jumlahSaiki* dengan 0x0A (dalam desimal adalah 10). Jika jumlahnya tidak sama maka program akan langsung meloncat ke *led2* tanpa menyalakan LED pertama. Namun, jika jumlahnya sama maka program akan terlebih dahulu menyalakan LED pertama (logic 0 pada port C bit 7) sebelum meloncat ke subrutin *led2*.

Pada subrutin *led2* dilakukan pengecekan jumlah lembar yang difotocopy dengan cara membandingkan *jumlahSaiki* dengan 0x14 (dalam desimal adalah 20). Jika jumlahnya tidak sama maka program akan langsung meloncat ke *led3* tanpa menyalakan LED kedua. Namun, jika jumlahnya sama maka program akan terlebih dahulu menyalakan LED kedua (logic 0 pada port C bit 6) sebelum meloncat ke subrutin *led3*.

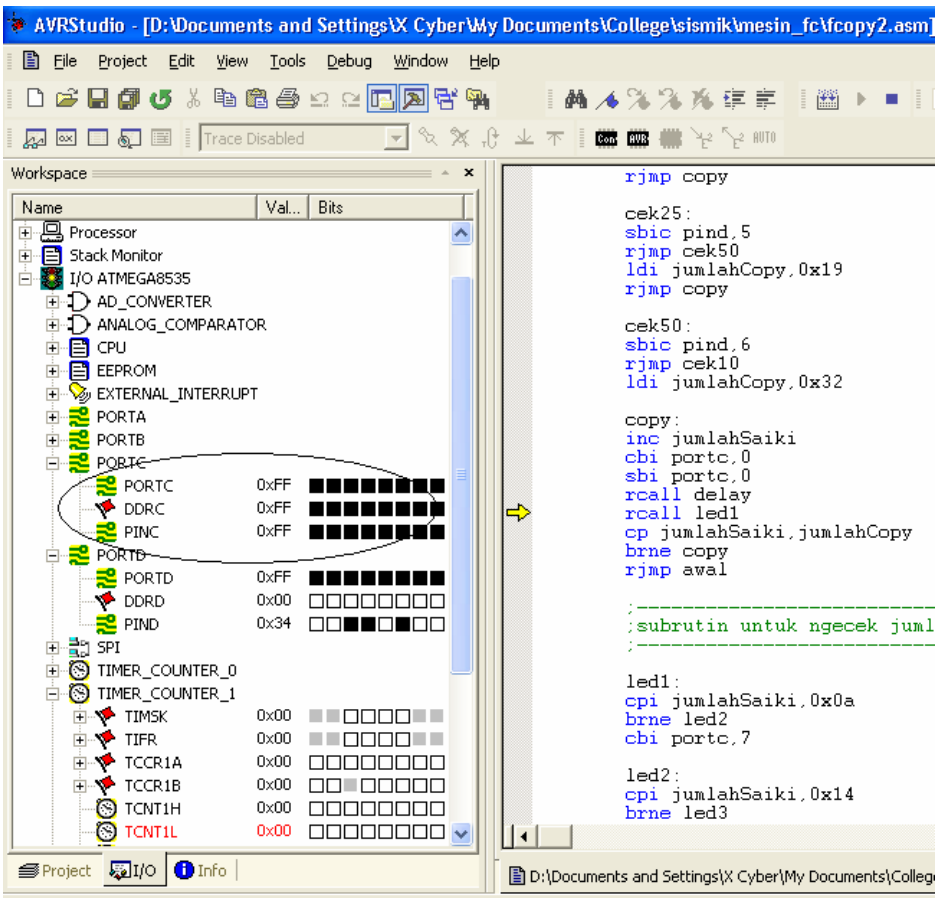
Pada subrutin *led3* kembali dilakukan pengecekan jumlah lembar yang difotocopy dengan cara membandingkan *jumlahSaiki* dengan 0x1E (dalam desimal adalah 30). Jika jumlahnya tidak sama maka program akan langsung meloncat ke *led4* tanpa menyalakan LED ketiga. Namun, jika jumlahnya sama maka program akan terlebih dahulu menyalakan LED ketiga (logic 0 pada port C bit 5) sebelum meloncat ke subrutin *led4*.

Begitupun pada subrutin *led4*, kembali dilakukan pengecekan jumlah lembar yang difotocopy dengan membandingkan *jumlahSaiki* dan 0x28 (dalam desimal adalah 40). Jika jumlahnya tidak sama maka program akan langsung meloncat ke *led5* tanpa menyalakan LED keempat. Namun, jika jumlahnya sama maka program akan terlebih dahulu menyalakan LED keempat (logic 0 pada port C bit 4) sebelum meloncat ke subrutin *led5*.

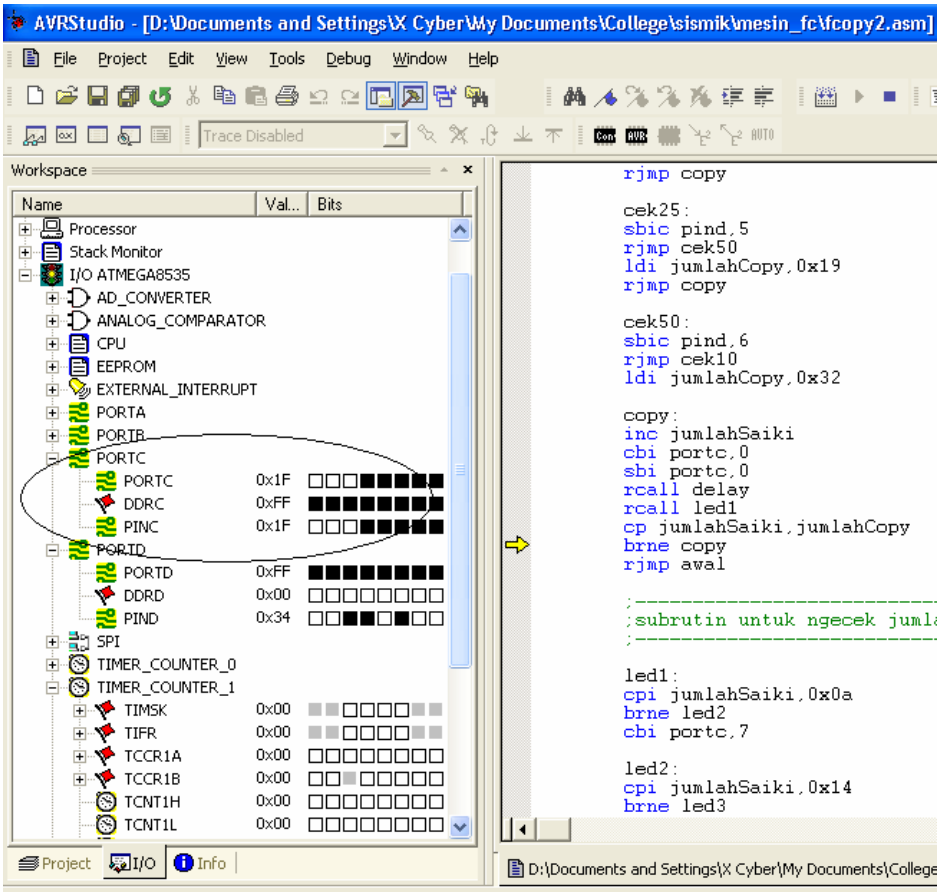
Subrutin *led5* juga melakukan pengecekan jumlah lembar yang difotocopy, yaitu membandingkan *jumlahSaiki* dengan 0x32 (dalam desimal adalah 50). Jika jumlahnya tidak sama maka program akan langsung keluar dari subrutin dan kembali melanjutkan proses selanjutnya tanpa menyalakan LED terakhir. Namun, jika jumlahnya sama maka program akan terlebih dahulu menyalakan LED kelima (logic 0 pada port C bit 3) sebelum keluar dan melanjutkan kembali proses berikutnya.

Berikut ini dapat dilihat hasil simulasi sebelum dan sesudah surutin *led1* dipanggil.

Sebelum subrutin LED dipanggil :



Setelah subrutin LED dipanggil :



Untuk mengatur delay antar tiap lembar fotokopi dilakukan dengan Timer. Pada dasarnya timer hanya menghitung siklus clock. Clock timer dapat sama dengan clock sistem (dari kristal, *RC oscilator* maupun clock eksternal) atau dapat juga diperlambat oleh *prescaler* terlebih dahulu. Ketika menggunakan *prescaler*, kita dapat menggunakan nilai timer yang lebih besar, tetapi presisinya akan menurun.

Prescaler dapat dipilih antara 8, 64, 256 dan 1024, tergantung pada sistem clocknya. Misalnya, sebuah AVR dengan *prescaler* dan frekuensi 8 MHz akan mampu menghitung sampai (0xFFFF + 1) x 1024 = 67.108.864 siklus clock, atau 8,388608 detik (diasumsikan bila menggunakan timer 16 bit). Pada contoh ini, *prescaler* akan meng-*increment* timer setiap 1024 siklus clock (0.000128 detik ≈ 0.125 µs).

Pada ATmega8535 terdapat dua jenis timer, yaitu Timer/Counter 8 bit dan Timer/Counter1 16 bit. Timer 8 bit sendiri ada dua, yaitu Timer/Counter0 8 bit dengan PWM (*Pulse Width Modulator*) serta Timer/Counter2 8 bit dengan PWM dan operasi asinkron. Timer 8 bit lebih sederhana bila dibandingkan dengan timer 16 bit. Clock timer akan meng-*counter* register Timer/Counter (TCNT0). Ketika timer overflow (dari 0xFF menjadi 0x00 lagi) maka *Overflow Interupt Flag* pada Timer 0 akan berlogic 1. Jika bit TOIE0 pada register TIMSK dan *global interupts* di-*enable*, maka mikrokontroller akan melompat ke vektor interupsi yang bersesuaian (dalam hal ini adalah vektor nomor 10 dengan alamat 0x0009).

Timer 16-bit sedikit lebih komplek. Perlu diingat, semua register 16 bit hanya dapat mengakses 1 byte (8 bit) dalam satu waktu. Untuk memastikan presisi waktu maka sebuah register temporary 16 bit digunakan ketika mengakses register timer. Ketika menulis *high-byte* (misalnya TCNT1H) maka data ditempatkan di register TEMP. Ketika *low-byte* ditulis, data dipindahkan ke register sesungguhnya secara simultan. Jadi, *high-byte* harus ditulis terlebih dahulu agar penulisan datanya benar. Sebaliknya ketika mengakses/membaca *low-byte*, *high-byte* juga dibaca dari TCNT1 secara simultan dan setelah itu baru dapat diakses. Maka, untuk menghasilkan pembacaan yang benar, *low-byte* harus diakses terlebih dahulu.

Pada perancangan keypad mesin fotocopy ini, timer yang digunakan adalah Timer1 16 bit. Kita akan memberi delay selama 0,25 detik. Pada simulasi penulis menge-set ATmega8535 dengan frekuensi 12 MHz. Jadi, 1 detik artinya adalah 12 juta *cycle* dan 0,25 detik berarti 3 juta *cycle*. Kita membutuhkan nilai *prescaler* yang besar. Penulis memilih untuk menggunakan *prescaler* 1024. Delay dilakukan dengan memanfaatkan overflow yang terjadi pada Timer 1. Jadi, setelah 0,25 detik Timer 1 akan mengalami overflow. Perhitungannya :

$$\frac{3.000.000}{1024} = 2929{,}6875$$

Maka, overflow akan terjadi setelah 2930 siklus-clock-timer. Kerja timer ini adalah dengan meng-*increment* isi register TCNT1 sampai terjadi overflow dari FFFF menjadi 0000. Oleh sebab itu, register TCNT1 diisi dengan -2930, atau dalam heksadesimal adalah F48E. Berikut cuplikan programnya.

```
;..... .....program di potong.....
.def data = r18
.equ waktu = 0xf48e

;.....program di potong.....
; subrutin delay 0,25 detik = 250.000 us = 3.000.000 cycle
;.....
delay:
ldi data,0x00                ;disable overflow interupt Timer1
out TIMSK,data
ldi data,high(waktu)         ;isi TCNT1 dengan 0xF48E
out TCNT1H,data
ldi data,low(waktu)
out TCNT1L,data
ldi data,0b00000101          ;jalankan Timer1 dengan prescaler : 1024
out TCCR1B,data

looptime:                    ;tunggu sampai Timer1 overflow flag set
in data,TIFR                 ;dapatkan status register TIFR
sbrs data,TOV1
rjmp looptime
ldi data,0b00000100          ;overflow flag dinolkan
out TIFR,data
ldi data,0x00                ;stop Timer1
out TCCR1B,data
ret
```

Pada program di atas overflow interupt dari Timer1 di-*disable* dengan memberikan logic 0 pada register TIMSK bit TOIE1. *Overflow interupt* di-*disable* agar program tidak meloncat ke vektor interupsi ketika terjadi overflow dan program dapat berjalan terus. Register *Timer Interrupt Mask Register* (TIMSK) digunakan untuk mengontrol interupsi yang valid dengan cara mengeset (memberi logic 1) pada bit-bit register tersebut. Berikut gambaran register TIMSK yang digunakan pada Timer1.

7	6	TICIE1	OCIE1A	OCIE1B	TOIE1	1	0

TIMSK

Keterangan:

- TOIE1 (*Timer Overflow Interrupt Enable Timer 1*) : jika bit ini di-set 1 dan *global interupt* di-*enable*, maka program akan meloncat ke vektor interupsi *Timer Overflow 1* ketika terjadi overflow pada Timer 1.
- OCIE1B (*Output Compare Interrupt Enable 1 B*) : jika bit ini di-set 1 dan *global interupt* di-*enable*, maka program akan meloncat ke vektor interupsi *Output Compare B* ketika nilai pada register TCNT1 sama dengan OCR1B.
- OCIE1A (*Output Compare Interrupt Enable 1 A*) : jika bit ini di-set 1 dan *global interupt* di-*enable*, maka program akan meloncat ke vektor interupsi *Output Compare A* ketika nilai pada register TCNT1 sama dengan OCR1A.
- TICIE1 (*Timer 1 Input Capture Interrupt Enable*) : jika bit ini di-set 1 dan *global interupt* di-*enable*, maka program akan meloncat ke vektor interupsi *Input Capture Interrupt* ketika terjadi input dari sinyal eksternal.
- Bit 0 dan 1 digunakan oleh Timer 0.
- Bit 6 dan 7 digunakan oleh Timer 2.

Setelah men-*disable* interupt dan mengisi register TCNT1 dengan 0xF48E selanjutnya adalah menjalankan Timer 1 dengan *prescaler* yang kita tentukan sebelumnya. Untuk mengatur nilai *prescaler* dapat dilakukan dengan mengkonfigurasi bit CS12, CS11 dan CS10 pada register TCCR1B.

ICNC1	ICES1	5	OCF1A	WGM12	CS12	CS11	CS10

TTCR1B

Berikut ini konfigurasi bit CS12, CS11 dan CS10 yang penulis peroleh dari http://www.avrbeginners.net.

CS12	CS11	CS10	Mode Description
0	0	0	Stop Timer/Counter 1
0	0	1	No Prescaler (Timer Clock = System Clock)
0	1	0	divide clock by 8
0	1	1	divide clock by 64
1	0	0	divide clock by 256
1	0	1	divide clock by 1024
1	1	0	increment timer 1 on T1 Pin falling edge
1	1	1	increment timer 1 on T1 Pin rising edge

Pada program yang penulis buat, bit CS12 dan CS10 diberi logic 1 sedangkan bit-bit lainnya diberi logic 0. Hal ini dilakukan untuk menjalankan Timer 1 dengan *prescaler* 1024. Sambil menunggu overflow, program akan terus mengulang loop dari subrutin *looptime*. Ketika terjadi overflow, maka bit TOV1 (*Timer/Counter 1 Overflow Flag*) pada register TIFR (*Timer Interrupt Flag*) akan berlogic 1. Sebagaimana TIMSK, TIFR juga digunakan untuk mengontrol interupsi. Seandainya saat terjadi overflow bit TOIE1 (register TIMSK) dan *global interupt* di-*enable*, maka program akan melompat ke vektor interusi nomor 7 pada alamat 0x0008. Tapi hal ini tidak dilakukan dan program tetap mengeksekusi perintah selanjutnya.

7	6	ICF1	OCF1A	OCF1B	TOV1	1	0

TIFR

Isi bit TOV1 inilah yang dijadikan tanda apakah delay sebesar 0,25 detik sudah tercapai atau belum. Jika bit ini bernilai 1 maka program akan keluar dari loop. Untuk me-nol-kan kembali TOV1 dapat dilakukan dengan cara memberi logic 1 pada bit tersebut. Sebenarnya, jika interupsi di-*enable* maka bit ini otomatis akan kembali bernilai 0 ketika program meloncat ke vektor interupsi. Namun, karena interupsi di-*disable* maka penge-nol-an bit TOV1 harus dilakukan oleh software/program. Setelah itu, Timer 1 distop dengan cara memberikan logic 0 pada CS12, CS11 dan CS10. Barulah program keluar dari subrutin *delay* dan melanjutkan proses fotocopy berikutnya.

Pada hasil simulasi sebelum subrutin *delay* dipanggil didapat data berikut:

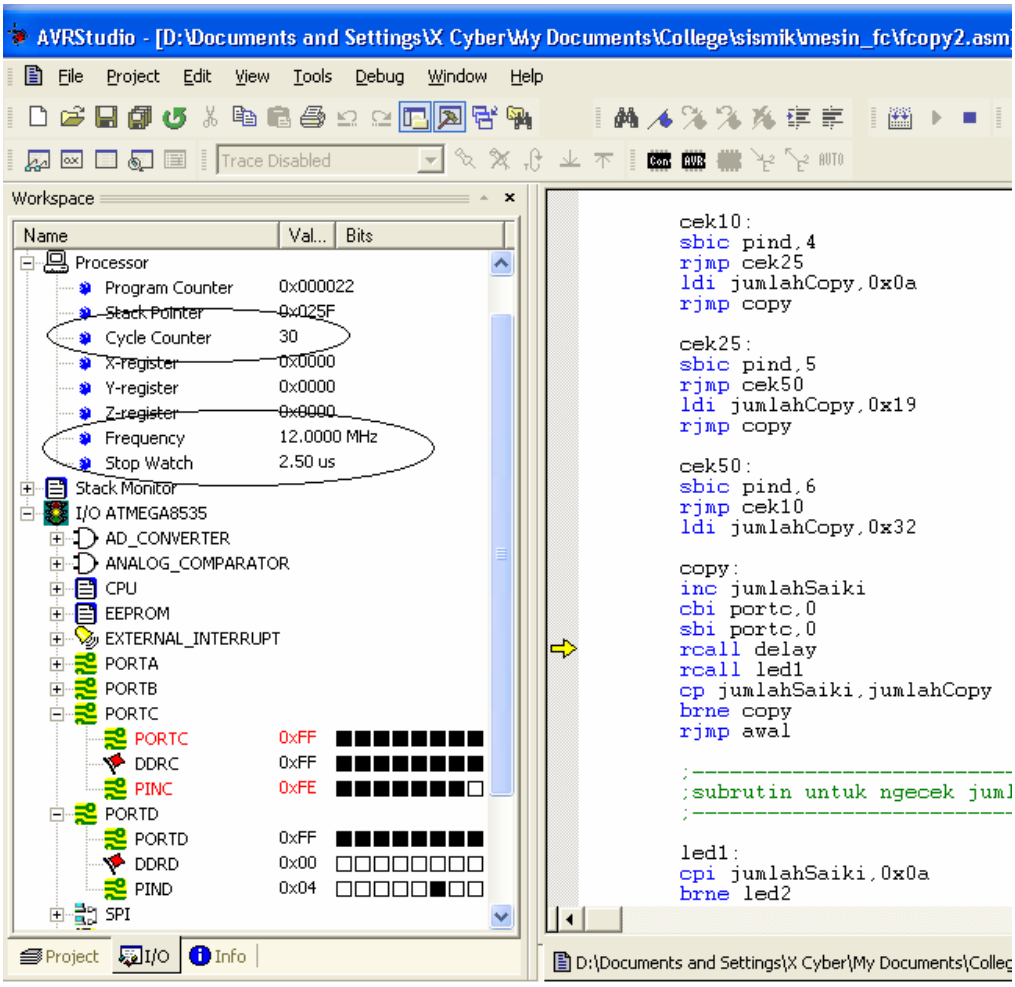
- Cycle Counter : 30
- Stop Watch : 2,50 us (microsecond)

Sedangkan sesudah subrutin *delay* dipanggil didapat data berikut:

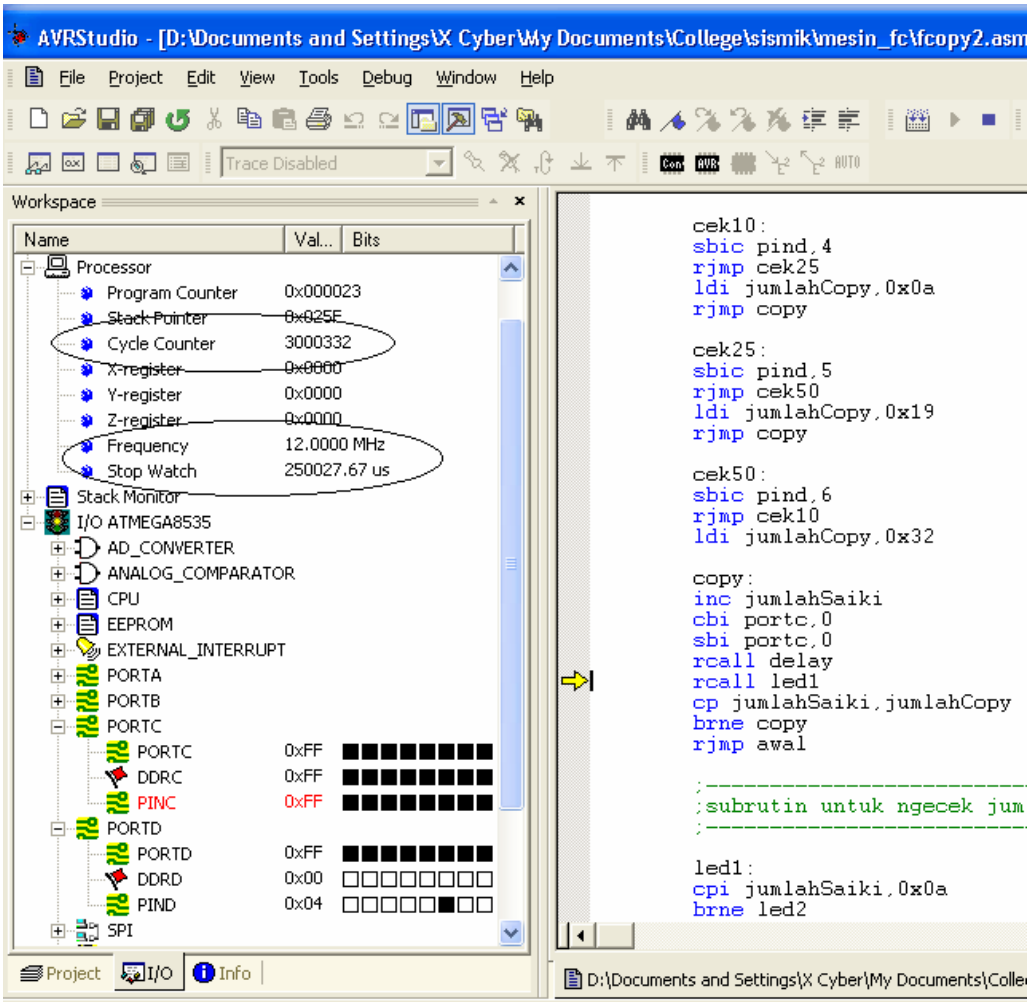
- Cycle Counter : 3.000.332
- Stop Watch : 250.027,67 us (microsecond)

Terlihat bahwa ada selisih 3.000.302 cycle (mendekati 3 juta cycle) antara sebelum dan sesudah subrutin *delay* dipanggil. Begitupun dengan Stop Watch, terdapat perbedaan 250.025,17 µs (mendekati 0,25 detik). Dengan demikian, delay sebesar 0,25 detik telah tercapai. Berikut hasil simulasi sebelum dan sesudah subrutin *delay* dipanggil.

Sebelum subrutin *delay* dipanggil :



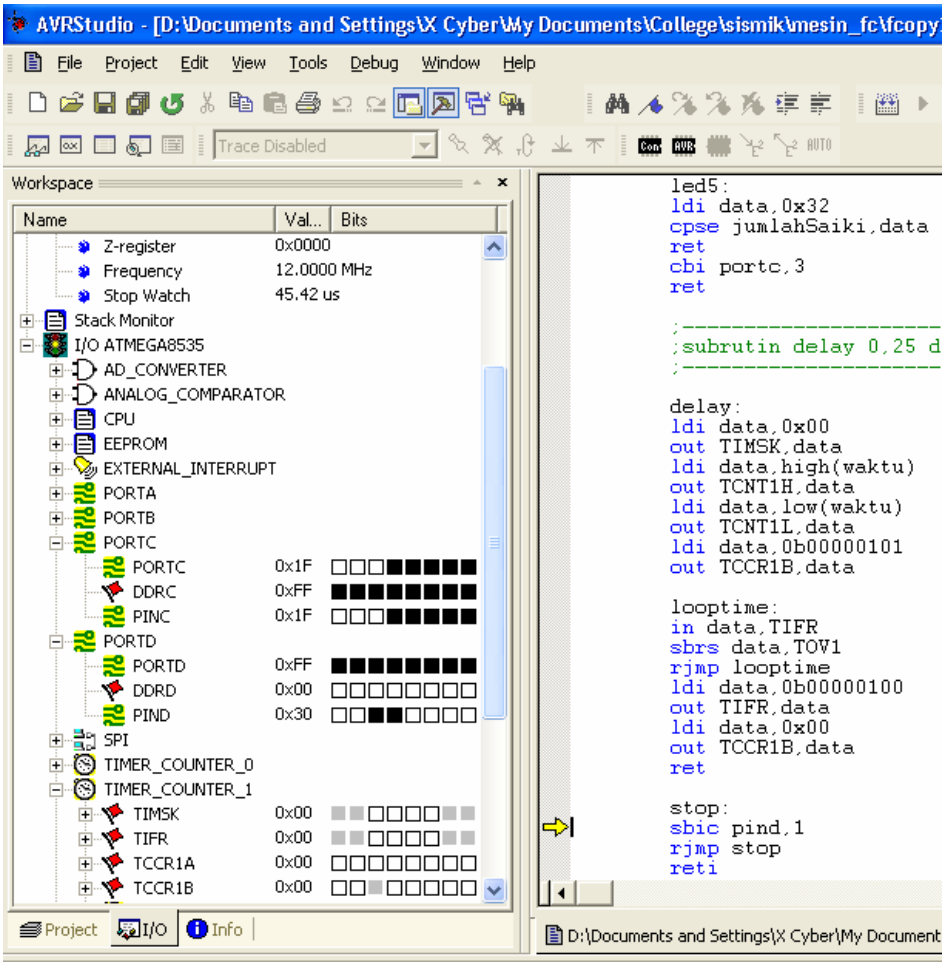
Setelah subrutin *delay* dipanggil :



Pada program yang penulis buat, interupsi eksternal 0 dari pin INT0 dapat terjadi kapanpun. Ketika pin ini ditekan maka program akan meloncat ke vektor interupsi kedua dengan alamat 0001 yang kemudian akan melakukan subrutin pelayanan interupsi, yaitu subrutin *stop*.

```
stop:                          ;subrutin eksternal interupt 0
sbic pind,1                    ;tunggu sampai pind 1 ditekan
rjmp stop
reti
```

Program akan terus mengulang loop stop sampai pin D bit 1 ditekan. Ketika pin ini ditekan maka program akan keluar dari subrutin pelayanan interupsi dan kembali melanjutkan ke proses selanjutnya. Berikut simulasi program ketika interupsi terjadi.



Semua program yang penulis jelaskan di atas akan terus dilakukan berulang-ulang oleh mikrokontroller, dalam hal ini Atmel AVR ATmega8535.

PENUTUP

Program yang penulis buat ini masih sangat sederhana. Dalam kenyataannya tentu saja keypad mesin fotocopy tidaklah sesederhana ini. Selain itu, program yang dibuat ini baru dijalankan sebatas simulasi, belum diujicoba langsung pada mikrokontrollernya (ATmega8535). Untuk itu, perlunya adanya ujicoba dan penyempurnaan-penyempurnaan di masa mendatang.

DAFTAR PUSTAKA

1. *Datasheet ATmega8535*, Atmel
2. Wardhana, Lingga. *Aplikasi Antarmuka Mikrokontroller AVR ATMega8535 dengan LCD 16x2*. Media Elektro Vol. 18 November 2005 - Januari 2006 (http://me.te.ugm.ac.id)
3. http://www.avrbeginners.net
4. http://www.atmel.com
5. http://papirus.te.ugm.ac.id