Arsitektur Dan Assembler AVR

January 14, 2019

AVR Data Memory ___

- AVR Data Memory terdiri dari GPR (General Purpose Register + I/O Register + internal data SRAM
  - 32 Byte GPR digunakan untuk menyimpan data secara umum
  - I/O Register digunakan untuk fungsi spesifik, meliputi Status Register, I/O port, ADC, komunikasi serial, timer dan peripheral lain
    - Ukuran: 8 bit
    - 64 Byte standard I/O register pada alamat $\$20 - \$5F$
    - Untuk AVR dengan pin di atas 32, ditambah extended I/O memory
  - Internal Memory SRAM digunakan untuk menyimpan data dan parameter oleh AVR programmer dan c compiler

---

AVR Data Memory ___

Table 2-1: Data Memory Size for AVR Chips

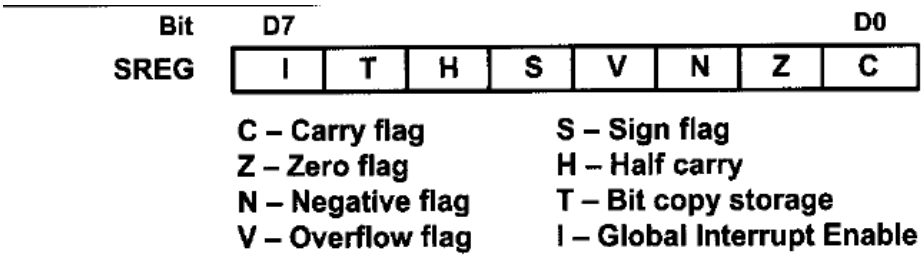
	Data Memory (Bytes)	=	I/O Registers (Bytes)	+	SRAM (Bytes)	+	General Purpose Register
ATtiny25	224		64		128		32
ATtiny85	608		64		512		32
ATmega8	1120		64		1024		32
ATmega16	1120		64		1024		32
ATmega32	2144		64		2048		32
ATmega128	4352		64+160		4096		32
ATmega2560	8704		64+416		8192		32

Extracted from http://www.atmel.com

AVR Data Memory

EEPROM Memory ___ * Memori EEPROM merupakan area memori read/write yang non-volatile * Biasanya digunakan untuk menyimpan data yang tidak boleh hilang, kalau sumber dayanya terputus * Alamat mulai dari 0x00 hingga nilai maksimal jumlah EEPROM * Jatang digunakan untuk penyimpanan variabel umum, karena lambat untuk dibaca dan ditulis

---



AVR Status Register

AVR Status Register ___ * Merupakan register 8-bit * Juga disebut Flag Register

- Bit C, Z, N, V,V,S, dan H disebut flag kondisional, yang mengindikasikan kondisi yang terjadi setelah sebuah instruksi dijalankan
- Setiap flag kondisional bisa digunakan untuk percabangan karena kondisi tertentu

---

Contoh Perintah Assembler ___

State the contents of RAM locations $212 to $216 after the following program is executed:

```
LDI   R16, 0x99   ;load R16 with value 0x99
STS   0x212, R16
LDI   R16, 0x85   ;load R16 with value 0x85
STS   0x213, R16
LDI   R16, 0x3F   ;load R16 with value 0x3F
STS   0x214, R16
LDI   R16, 0x63   ;load R16 with value 0x63
STS   0x215, R16
LDI   R16, 0x12   ;load R16 with value 0x12
STS   0x216, R16
```

Jawab ___

Solution:

After the execution of `STS 0x212, R16` data memory location $212 has value 0x99;
after the execution of `STS 0x213, R16` data memory location $213 has value 0x85;
after the execution of `STS 0x214, R16` data memory location $214 has value 0x3F;
after the execution of `STS 0x215, R16` data memory location $215 has value 0x63;
and so on, as shown in the chart.

Address	Data
$212	0x99
$213	0x85
$214	0x3F
$215	0x63
$216	0x12

I/O Register ___

Address		Name
Mem.	I/O	
$20	$00	TWBR
$21	$01	TWSR
$22	$02	TWAR
$23	$03	TWDR
$24	$04	ADCL
$25	$05	ADCH
$26	$06	ADCSRA
$27	$07	ADMUX
$28	$08	ACSR
$29	$09	UBRRL
$2A	$0A	UCSRB
$2B	$0B	UCSRA
$2C	$0C	UDR
$2D	$0D	SPCR
$2E	$0E	SPSR
$2F	$0F	SPDR
$30	$10	PIND
$31	$11	DDRD
$32	$12	PORTD
$33	$13	PINC
$34	$14	DDRC
$35	$15	PORTC

Address		Name
Mem.	I/O	
$36	$16	PINB
$37	$17	DDRB
$38	$18	PORTB
$39	$19	PINA
$3A	$1A	DDRA
$3B	$1B	PORTA
$3C	$1C	EECR
$3D	$1D	EEDR
$3E	$1E	EEARL
$3F	$1F	EEARH
$40	$20	UBRRC
		UBRRH
$41	$21	WDTCR
$42	$22	ASSR
$43	$23	OCR2
$44	$24	TCNT2
$45	$25	TCCR2
$46	$26	ICR1L
$47	$27	ICR1H
$48	$28	OCR1BL
$49	$29	OCR1BH
$4A	$2A	OCR1AL

Address		Name
Mem.	I/O	
$4B	$2B	OCR1AH
$4C	$2C	TCNT1L
$4D	$2D	TCNT1H
$4E	$2E	TCCR1B
$4F	$2F	TCCR1A
$50	$30	SFIOR
$51	$31	OCDR
		OSCCAL
$52	$32	TCNT0
$53	$33	TCCR0
$54	$34	MCUCSR
$55	$35	MCUCR
$56	$36	TWCR
$57	$37	SPMCR
$58	$38	TIFR
$59	$39	TIMSK
$5A	$3A	GIFR
$5B	$3B	GICR
$5C	$3C	OCR0
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Perintah IN dan OUT untuk manipulasi I/O register ___ Contoh berikut, perintah untuk menambahkan isi PIND ke PINB

```
IN    R1,PIND      ;load R1 with PIND
IN    R2,PINB      ;load R2 with PINB
ADD   R1, R2       ;R1 = R1 + R2
STS   0x300, R1    ;store R1 to data space location $300
```

Contoh:, copy nilai PIND ke PORTA:

```
IN    R0, PIND     ;load R20 with the contents of I/O reg PIND
OUT   PORTA, R0    ;out R20 to PORTA
```

1 Pemrograman AVR dengan Assembler

- CPU hanya mengerti bahasa mesin, di pihak lain manusia kesulitan untuk mengingat bahasa mesin yang direpresentasikan dalam bilangan biner

- Bahasa assembler dibuat untuk menjembatani hal tersebut dengan menyediakan perintah yang disebut mnemonic, yang mudah diingat oleh manusia, dan bisa diterjemahkan langsung ke dalam bahasa mesin oleh tools yang disebut assembler.
- Bahasa assembler disebut pemrograman low level karena mengakses langsung struktur internal dari CPU

1.1 Struktur Bahasa Asembler

- Kode Bahasa Asembler terdiri dari sejumlah statement, yang meliputi:
  - directive: perintah yang merupakan petunjuk bagi assembler terkait penterjemahan kode ke dalam bahasa mesin
  - perintah assembler: perintah yang akan diterjemahkan ke dalam bahasa mesin oleh assembler, berisi mnemonic + operands
- Struktur kode assembler meliputi 4 kolom (dalam kurung adalah opsional):

```
[label:]   mnemonic/directive   [operands]   [;komentar]
```

Contoh kode assembler: ___

```
        ;ini adalah komentar, diawali dengan ";"
        ;kode berikut membuat variabel dengan nama SUM
        ;yang merupakan nama untuk lokasi di SRAM pada
        ;alamat 0x300

        .EQU SUM = 0x300 ;directive, diawali dengan titik
                         ;petunjuk bagi asembler
                         ;bahwa SUM adalah lokasi pada alamat
                         ;0x300 di SRAM
        .ORG 00          ;petunjuk bagi assembler untuk mulai
                         ;pada alamat 00 (Program Counter
                         ;diset 0

        LDI R16, 0x25    ;mnemonic, isi R16 dengan 0x25
        LDI R17, $34     ;isi R17 dengan nilai 0x34
        LDI R18, 0b00110001 ; isi R18 dengan 0x31
        ADD R16, R17     ;Tambahkan isi R16 dengan
                         ;nilai pada  R17
        ADD R16, R18     ;tambahkan isi R16 dengan
                         ;nilai pada R18
        STS SUM, R16     ;copykan nilai R16 ke alamat 0x300
LOOP1:  JMP LOOP1        ;ini adalah loop terus menerus,
                         ;perintah percabangan (contoh JMP)
                         ;diikuti label tempat perintah
                         ;selanjutnya dijalankan
```

1.1.1 Directive

Berikut adalah directive pada pemrograman assembler

Directive	Operand	Penggunaan	Contoh
.INCLUDE	“nama file”	menambahkan data teks	.INCLUDE “m328def.inc”
.DEVICE	tipe mk	mendefinisikan tipe mk	.DEVICE ATMEGA328

```
In [1]: ### Operand Assembler

In [2]: ### Branch, dan loop

In [3]: ### Fungsi assembler

In [ ]: ### Operator

In [ ]: ### Operasi bit

In [ ]: ### Operasi word (16 bit)

In [ ]:
```