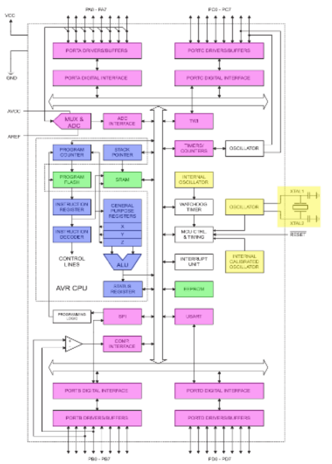
به نام خدا

آشنایی با زبان اسمبلی AVR

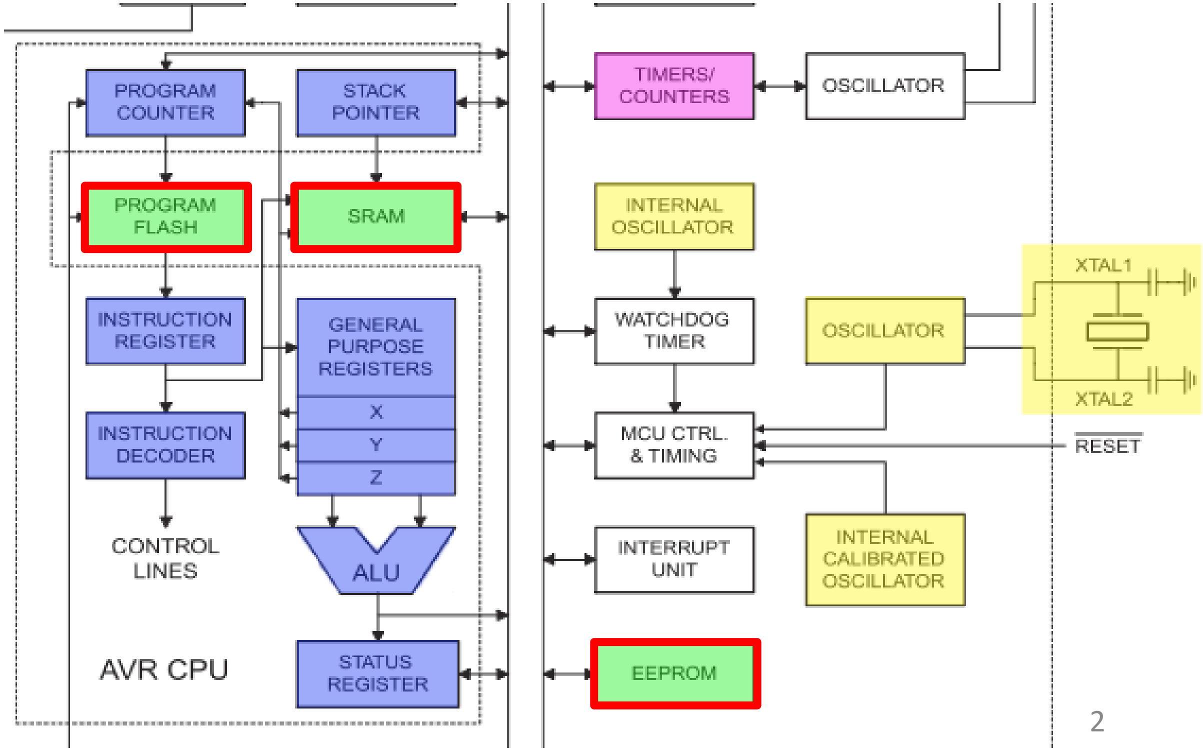
کار با حافظه

Dr. Aref Karimiafshar
A.karimiafshar@ec.iut.ac.ir

کار با حافظه

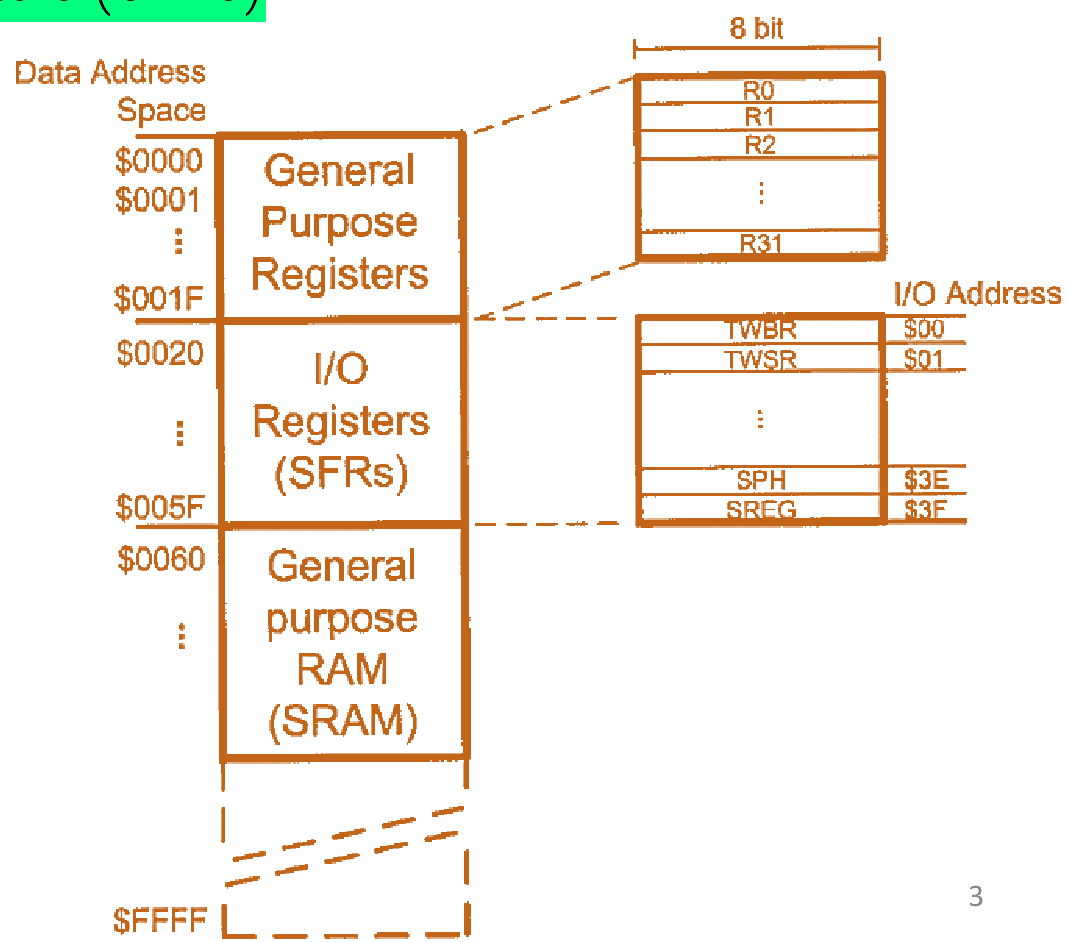


- Two kinds of memory space in AVR:
  - Code memory space
    - Stores our program
  - Data memory space
    - Stores data



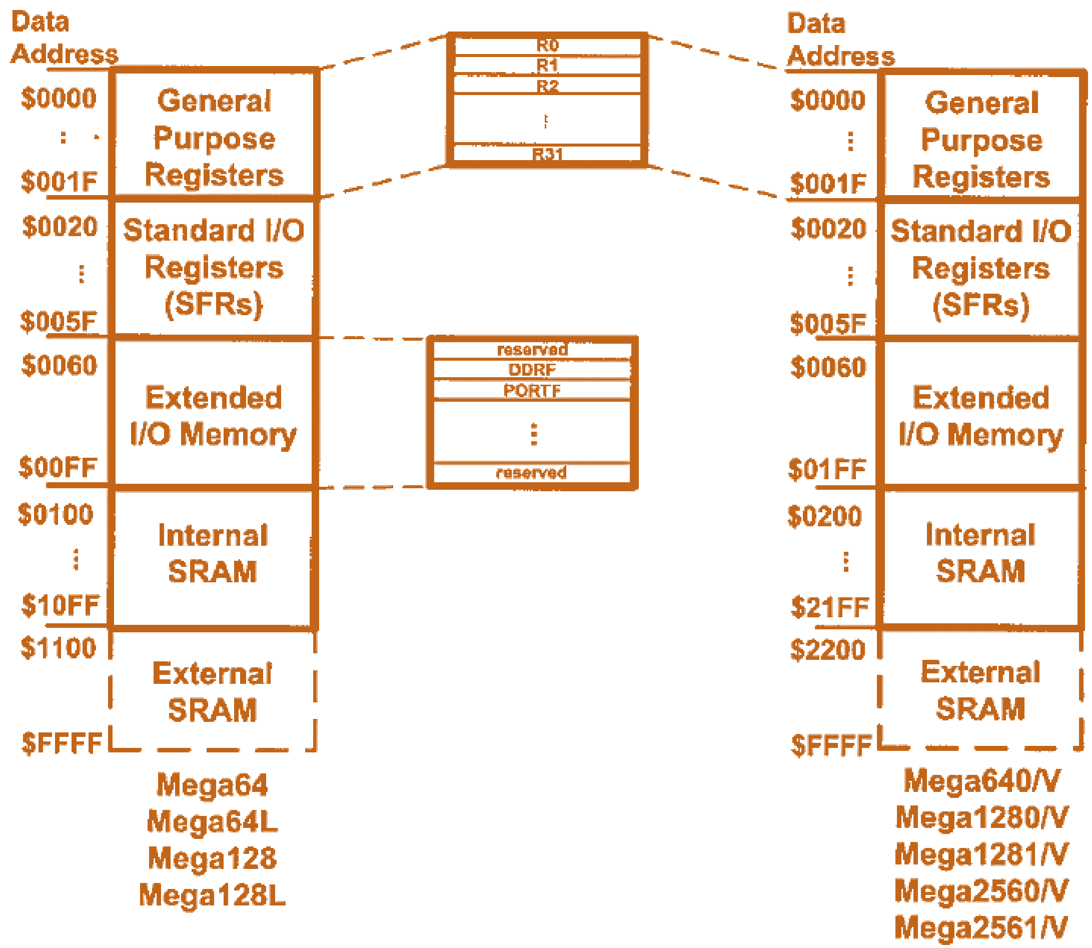
حافظه داده

- Data memory space
  - General Purpose Registers (GPRs)
    - 32x8 registers
  - I/O memory
    - Status register
    - Timer
    - Serial communication
    - I/O ports
    - ADC
    - …
  - Internal data SRAM



I/O حافظه

- I/O memory
  - Special Function Registers (SFRs)
  - Number of locations in data mem. set aside for I/O mem depends on the pin numbers and peripheral functions
  - However, all AVRs have at least 64 bytes of I/O mem
    - Called standard I/O mem
  - AVRs with more than 32 I/O pins
    - Have an extended I/O mem



SRAM حافظه

- Internal data SRAM
  - Storing data and parameters
    - Called scratch pad
  - Each location of SRAM can be accessed directly by its address
  - Each location is 8-bit wide
  - Size of SRAM can vary from chip to chip

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

LDS – Load Direct from Data Space

- Loads one byte from the data space to a register.

(i) Rd ← (k)

	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	$0 \le d \le 31$, $0 \le k \le 65535$	PC ← PC + 2

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula

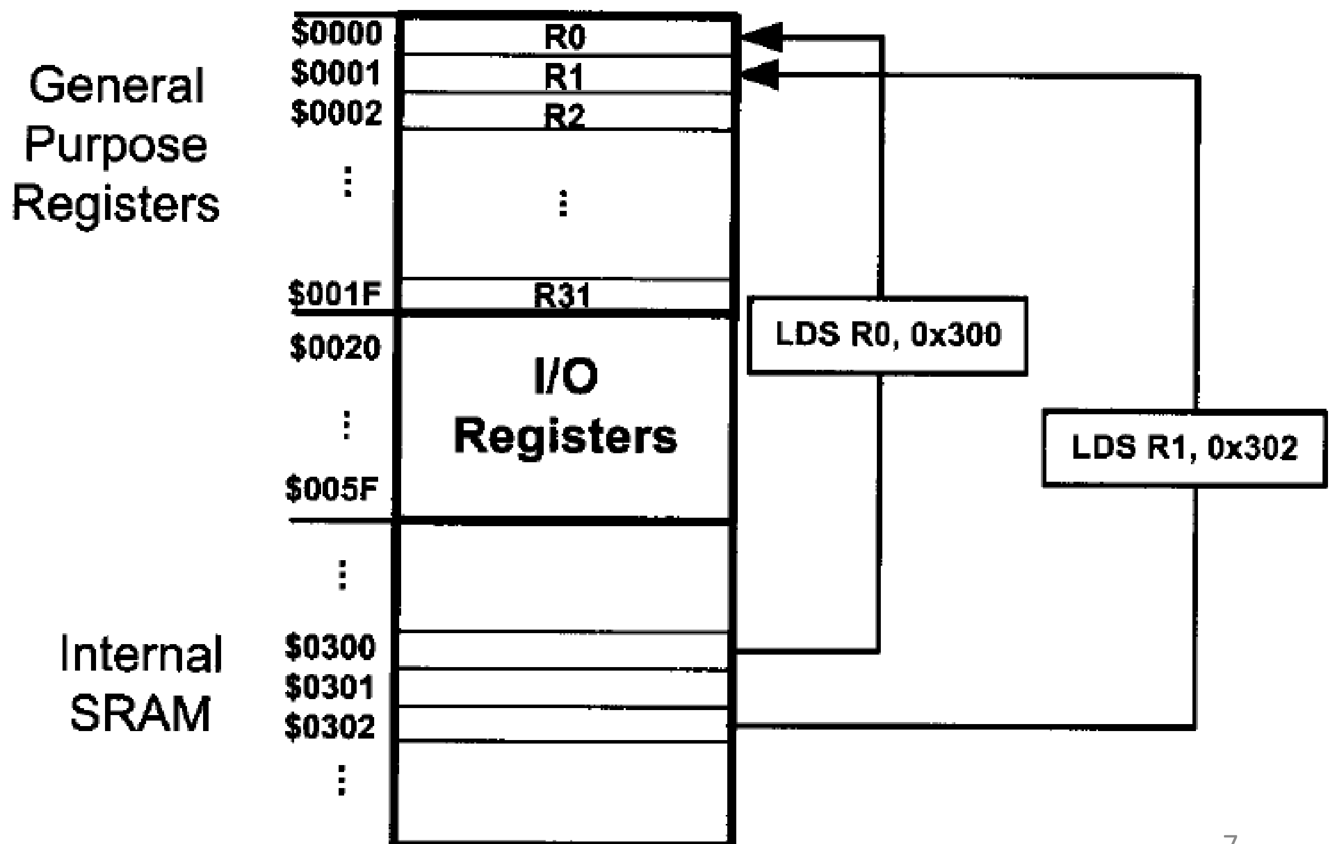
I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 2 (4 bytes)

Cycles 2

LDS – Example

```
LDS    R0, 0x300    ;R0 = the contents of location 0x300
LDS    R1, 0x302    ;R1 = the contents of location 0x302
```



STS – Store Direct to Data Space

- Stores one byte from a Register to the data space.

(i) (k) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	STS k,Rr	$0 \le r \le 31$, $0 \le k \le 65535$	PC ← PC + 2

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 2 (4 bytes)

Cycles 2

I/O Registers

- Each location in I/O memory has two addresses
  - I/O address
  - Data memory address

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

IN - Load an I/O Location to Register

- Loads data from the I/O Space (Ports, Timers, Configuration Registers, etc.) into register Rd in the Register File.

(i) Rd ← I/O(A)

	Syntax:	Operands:	Program Counter:
(i)	IN Rd,A	$0 \le d \le 31$, $0 \le A \le 63$	PC ← PC + 1

16-bit Opcode:

1011	0AAd	dddd	AAAA

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	1

IN - Load an I/O Location to Register

To work with the I/O registers more easily, we can use their names instead of their I/O addresses. For example, the following instruction loads R19 with the contents of PIND:

```
IN R19,PIND        ;load R19 with PIND
```

The following program adds the contents of PIND to PINB, and stores the result in location 0x300 of the data memory:

```
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
```

IN vs. LDS

- IN is faster than LDS
  - IN lasts 1 MC, LDS lasts 2 MC
- IN occupies less memory
  - IN is 2-Byte instruction, LDS is 4-Byte instruction
- When we use IN
  - We can use the names of I/O registers
- IN is available in all AVRs, LDS implemented in some

OUT – Store Register to I/O Location

- Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers, etc.).

(i) $I/O(A) \leftarrow Rr$

	Syntax:	Operands:	Program Counter:
(i)	OUT A,Rr	$0 \le r \le 31$, $0 \le A \le 63$	$PC \leftarrow PC + 1$

16-bit Opcode:

1011	1AAr	rrrr	AAAA

Status Register (SREG) and Boolean Formula

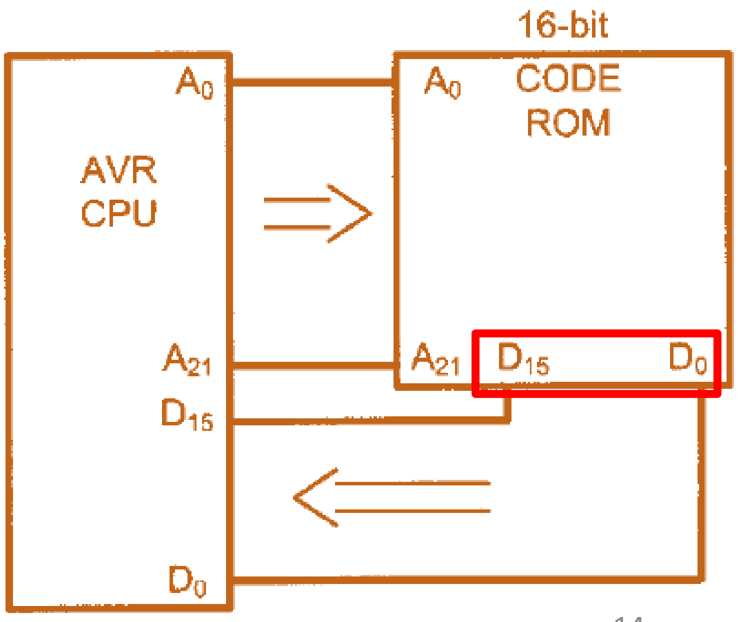
I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words 1 (2 bytes)

Cycles 1

Program ROM Space

- Program counter
  - Point to the address of next instruction
- Program counter width
  - The wider program counter, the wider address space
- In AVR microcontroller
  - Each Flash mem. location is 2 bytes
  - Example: ATmega32 → 32KB Flash
    - Organized as 16Kx16
    - PC → 14 bits wide



Program ROM Space

AVR On-chip ROM Size and Address Space

	On-chip Code ROM (Bytes)	Code Address Range (Hex)	ROM Organization
ATtiny25	2K	00000–003FF	1K × 2 bytes
ATmega8	8K	00000–00FFF	4K × 2 bytes
ATmega32	32K	00000–03FFF	16K × 2 bytes
ATmega64	64K	00000–07FFF	32K × 2 bytes
ATmega128	128K	00000–0FFFF	64K × 2 bytes
ATmega256	256K	00000–1FFFF	128K × 2 bytes

- In AVR microcontroller
  - PC can be up to 22 bits wide
  - Access program address 000000 to $3FFFFF
    - Total of 4M locations ---> 8M bytes on-chip ROM

Assembler Directives

- Give directions to the assembler
- Directives help us
  - Develop our program easier
  - Make our program legible (more readable)

- Directives
  - .EQU
  - .SET
  - .ORG
  - …

```
.EQU COUNTER = 0x00
.EQU PORTB = 0x18
LDI  R16, COUNTER
OUT  PORTB, R16
```

Assembler Directives

.EQU

- Define a constant value
- Does not set aside storage for a data item
- Associate a constant number with a label

```
.EQU  COUNT = 0x25
      ...    ....
      LDI    R21, COUNT          ;R21 = 0x25
```

.SET

- Define a constant value
- Like .EQU; difference → may be reassigned later

Assembler Directives

.ORG
- Indicate the beginning of the address

.INCLUDE
- Add the contents of a file to our program

- When you want to use ATmega32
  - You must write the following at the beginning of your program

```
.INCLUDE "M32DEF.INC"
```

Assembler Directives

.DB

- Allocate program memory in byte-sized chunks
- The number can be:
  - Binary `.DB 0xb0101`
  - Decimal `.DB 28`
  - Hex `.DB 0xA`
  - ASCII `.DB 's'`

.DW

- Allocate program memory in word-sized chunks

Assembler Directives

.ESEG

- Variable will be located in EEPROM

```
.ESEG
    .DB 0b0101
    .DB 0xE
```

.CSEG

- Variable will be located in Code memory

```
.CSEG
    .DB 0b0101
    .DB 0xE
```

Assembler Directives

.DSEG

– Variable will be located in SRAM

```
.DSEG
    .DB 0b0101
    .DB 0xE
```

.EXIT

– Stop the execution

Assembly Language Ins. Format

- Assembly language instructions consist of four fields:

```
[label:]   mnemonic   [operands]   [;comment]
```

```
         .EQU  SUM   = 0x300      ;SRAM loc $300 for SUM

         .ORG 00                  ;start at address 0
         LDI R16, 0x25            ;R16 = 0x25
         LDI R17, $34             ;R17 = 0x34
         LDI R18, 0b00110001      ;R18 = 0x31
         ADD R16, R17             ;add R17 to R16
         ADD R16, R18             ;add R18 to R16
         LDI R17, 11              ;R17 = 0x0B
         ADD R16, R17             ;add R17 to R16
         STS SUM, R16             ;save the SUM in loc $300
   HERE: JMP HERE                 ;stay here forever
```

EEPROM حافظه

- Data in SRAM will be lost if power is disconnected
- EEPROM can save stored data even if power is cut off
- Accessing EEPROM in AVR
  - Three I/O registers, directly related to EEPROM
    - EECR (EEPROM Control Register)
    - EEDR (EEPROM Data Register)
    - EEARH : EEARL (EEPROM Address Register High-Low)

Size of EEPROM Memory in ATmega Family

Chip	Bytes	Chip	Bytes	Chip	Bytes
ATmega8	512	ATmega16	512	ATmega32	1024
ATmega64	2048	ATmega128	4096	ATmega256RZ	4096
ATmega640	4096	ATmega1280	4096	ATmega2560	4096

رجیسترهای کار با EEPROM

- EEDR (EEPROM Data Register)
  - To write data to EEPROM, you have to write it to EEDR
    - Then transfer it to EEPROM
  - To read data from EEPROM, you have to read from EEDR

- EEARH : EEARL (EEPROM Address Register High-Low)
  - Together make a 16-bit reg. to address each location in EEPROM
    - 10 bits are used in Atmega32

Bit	15	14	13	12	11	10	9	8
EEARH	-	-	-	-	-	-	EEAR9	EEAR8
EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
Bit	7	6	5	4	3	2	1	0

رجیسترهای کار با EEPROM

- EECR (EEPROM Control Register)
  - Select the kind of operation to perform
    - Start
    - Read
    - Write

EECR	-	-	-	-	EERIE	EEMWE	EEWE	EERE
Bit	7	6	5	4	3	2	1	0

  - EERE→ EEPROM Read Enable
  - EEWE→ EEPROM Write Enable

نوشتن EEPROM

To write on EEPROM the following steps should be followed. Notice that steps 2 and 3 are optional, and the order of the steps is not important. Also note that you cannot do anything between step 4 and step 5 because the hardware clears the EEMWE bit to zero after four clock cycles.

1. Wait until EEWE becomes zero.
2. Write new EEPROM address to EEAR (optional).
3. Write new EEPROM data to EEDR (optional).
4. Set the EEMWE bit to one (in EECR register).
5. Within four clock cycles after setting EEMWE, set EEWE to one.

نوشتن EEPROM

Write an AVR program to store 'G' into location 0x005F of EEPROM .

Solution:

```
.INCLUDE "M16DEF.INC"

WAIT:                   ;wait for last write to finish
SBIC   EECR,EEWE        ;check EEWE to see if last write is finished
RJMP   WAIT             ;wait more
LDI    R18,0            ;load high byte of address to R18
LDI    R17,0x5F         ;load low byte of address to R17
OUT    EEARH, R18       ;load high byte of address to EEARH
OUT    EEARL, R17       ;load low byte of address to EEARL
LDI    R16,'G'          ;load 'G' to R16
OUT    EEDR,R16         ;load R16 to EEPROM Data Register
SBI    EECR,EEMWE       ;set Master Write Enable to one
SBI    EECR,EEWE        ;set Write Enable to one
```

EEPROM خواندن از

To read from EEPROM the following steps should be taken. Note that step 2 is optional.

1. Wait until EEWE becomes zero.
2. Write new EEPROM address to EEAR (optional).
3. Set the EERE bit to one.
4. Read EEPROM data from EEDR.

خواندن از EEPROM

Write an AVR program to read the content of location 0x005F of EEPROM into PORTB.

Solution:

```
.INCLUDE "M16DEF.INC"
        LDI    R16,0xFF
        OUT    DDRB,R16
WAIT:                        ;wait for last write to finish
        SBIC   EECR,EEWE     ;check EEWE to see if last write is finished
        RJMP   WAIT          ;wait more
        LDI    R18,0         ;load high byte of address to R18
        LDI    R17,0x5F      ;load low byte of address to R17
        OUT    EEARH, R18    ;load high byte of address to EEARH
        OUT    EEARL, R17    ;load low byte of address to EEARL
        SBI    EECR,EERE     ;set Read Enable to one
        IN     R16,EEDR      ;load EEPROM Data Register to R16
        OUT    PORTB,R16     ;out R16 to PORTB
```

معماری RISC

RISC processors have a fixed instruction size. In a CISC microcontroller such as the 8051, instructions can be 1, 2, or even 3 bytes. For example, look at the following instructions in the 8051:

```
CLR   C                        ;clear Carry flag, a 1-byte instruction
ADD   Accumulator, #mybyte     ;a 2-byte instruction
LJMP  target_address           ;a 3-byte instruction
```

RISC uses load/store architecture. In CISC microprocessors, data can be manipulated while it is still in memory. For example, in instructions such as "ADD Reg, Memory", the microprocessor must bring the contents of the external memory location into the CPU, add it to the contents of the register, then move the result back to the external memory location. The problem is there might be a delay in accessing the data from external memory.

پایان

موفق و پیروز باشید