بسم الله الرحمن الرحیم

میکرو کنترلرهای AVR

معماری AVR و برنامه نویسی اسمبلی

دانشکده برق و رباتیک

دانشگاه صنعتی شاهرود

حسین خسروی

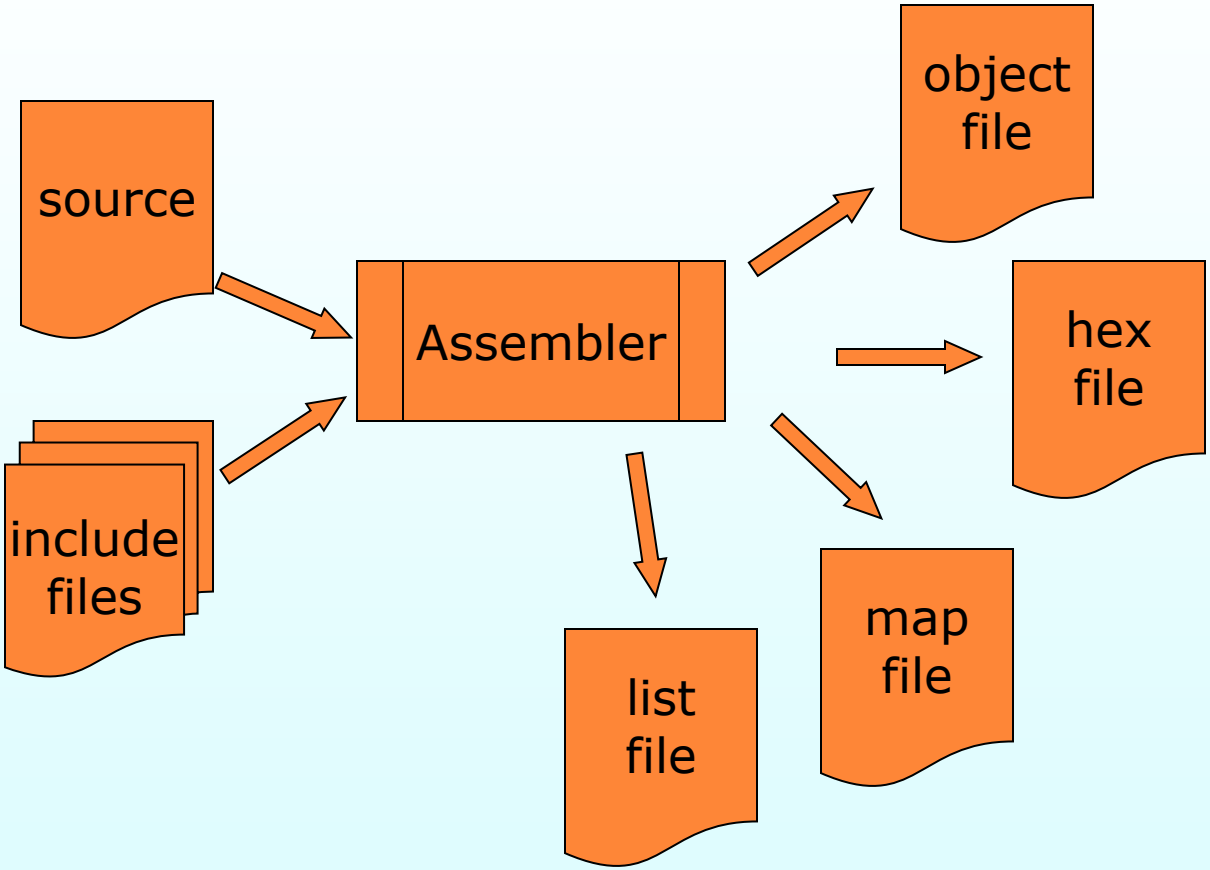
۱۳۹۰-۹۱

Assembly introduction

- Low-level programming language
- Architecture dependent (e.g. x86, 8051, AVR…)
- Between C and machine code – compact,
- Application: mainly small embedded systems (pl. PIC, AVR)
- For large projects: asm is expensive, inflexible, hard to manage; C compilers are well-optimized
  - Low-level routines
  - Computations intensive tasks (mathematics, graphics)
  - reverse engineering

Assemblers

- ترجمه کد اسمبلی به کد ماشین



ثبات‌های همه منظوره در AVR

	R0	$00	
	R1	$01	
	R2	$02	
	...		
	R13	$0D	
General	R14	$0E	
Purpose	R15	$0F	
Working	R16	$10	
Registers	R17	$11	
	...		
	R26	$1A	X-register Low Byte
	R27	$1B	X-register High Byte
	R28	$1C	Y-register Low Byte
	R29	$1D	Y-register High Byte
	R30	$1E	Z-register Low Byte
	R31	$1F	Z-register High Byte

AVR assembly - registers

- RISC instruction set, load/store architecture: Registers:
  - 32, 8 bit wide (R0…R31)
  - Always at address 0x00 – 0x1F
  - All operations are done through registers
  - Last six serves as register pairs
    - Implement 3 16-bit registers (X, Y, Z)

AVR assembly - instructions

mnemonic

arguments (operands)

```
ldi R31 ,  0xA5   ; 10100101
out PORTC, R30   ; port write
```

comment

!!!!!

AVR assembly - instructions

instruction arguments

```
ldi R31 ,  0xA5   ; 10100101
out PORTC, R30    ; port write
```

SREG

Mnemo.	Operands	Description	Operation	Flags	#Clk
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V	2
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,H	1

دستور LDI و ADD

- کپی کردن یک داده ۸ بیتی درون یک ثبات همه منظوره

```
LDI Rd, K          ; Immediate load (16<d<32)
LDI R20,0x25       ;R20 = 0x25
LDI R31, 0x87
LDI R25, 0x123     ; invalid value
LDI R10, 0x20      ; invalid register
```

- جمع کردن

```
ADD Rd, Rr    ;Rd = Rd + Rr
```

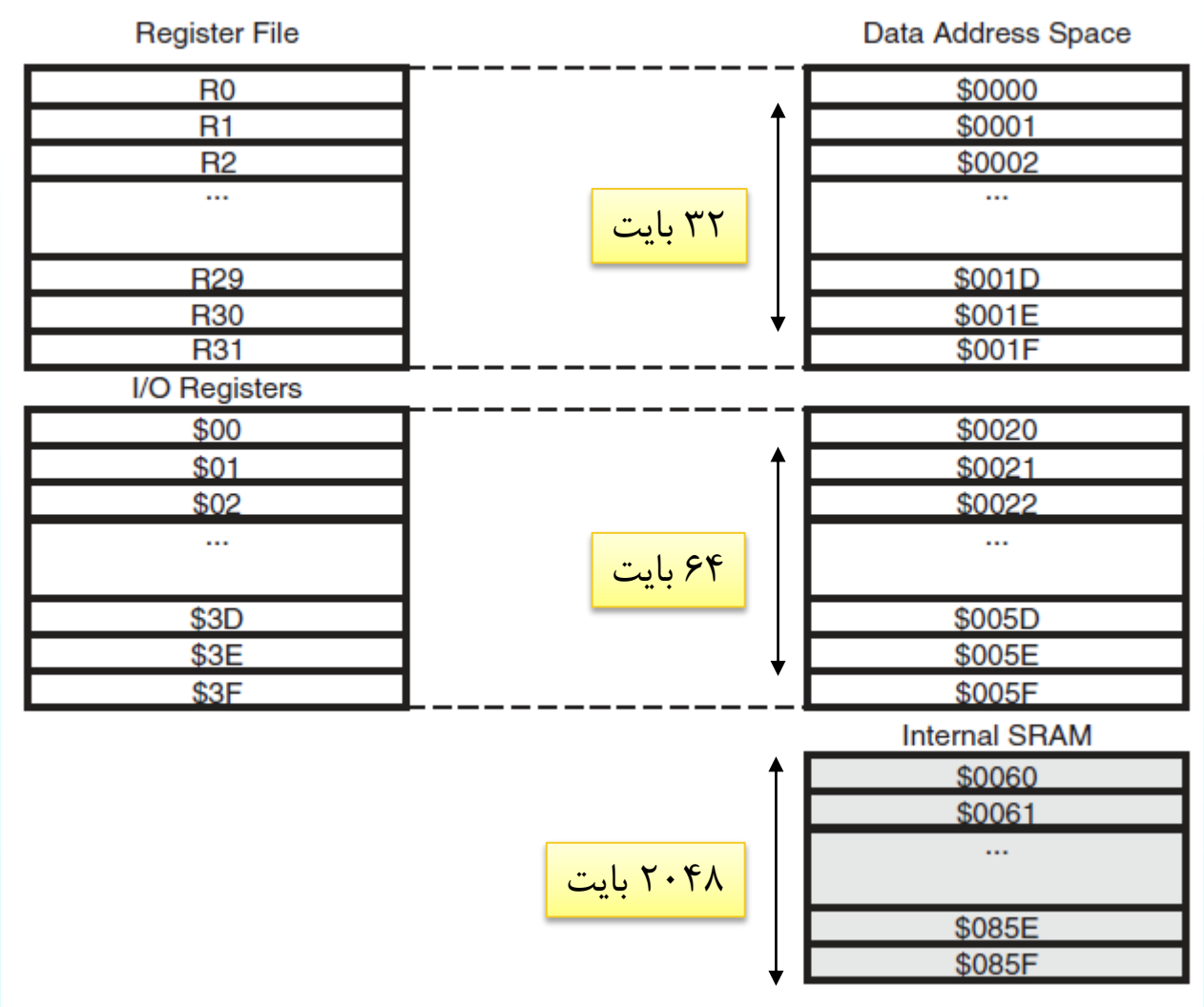
  - جمع دو عدد 0x25 و 0x34

```
LDI R16,0x25
LDI R17,0x34
ADD R16,R17
```

حافظه I/O (SFRs)

- ثباتهای با کارکرد خاص
  - ثبات وضعیت
  - تایمرها
  - ارتباط سریال
  - پورتهای I/O
  - ADC
  - ...
- هر AVR حداقل ۶۴ بایت فضای حافظه I/O دارد

نگاشت حافظه داده



مجموعا ۲۱۴۴ بایت

دسترسی به محتوای حافظه RAM و SFR

- دو دستور LDS و STS برای کپی کردن اطلاعات از/به RAM کاربرد دارند:
  - LDS (LoaD from data Space)
  - STS (Store to data Space)

```
LDS Rd, k; Rd = contents of location k (d: 0-31)
STS k, Rd; contents of location k = Rd (d: 0-31)
; k is an address between 0x000-0xffff
```

مثال: جمع محتوای خانه های 0x300 و 0x302 حافظه و ذخیره در آدرس 0x38 (پورت B)

```
LDS R0,0x300
LDS R0,0x302
ADD R1, R0
STS 0x38,R1
```

مثال: محتوای ثباتهای R20 و R21 و خانه 0x120 بعد از اجرای برنامه؟

```
LDI R20, 5
LDI R21, 2
ADD R20, R21
ADD R20, R21
STS 0x120, R20
```

دسترسی به IO با دستورات IN و OUT

- خانه حافظه I/O دو آدرس دارند: آدرس حافظه و آدرس I/O

I/O Address = Mem. Address – 0x20

- برخلاف LDI و STS دستورات IN و OUT با آدرسهای I/O کار می کنند:

```
IN Rd, k; Rd = contents of location k (d: 0-31)
STS k, Rd; contents of location k = Rd (d: 0-31)
; k is an address between 0x000-0xffff
```

مثال: برای دسترسی به (داده های ورودی پورت B) آدرس 0x16 رابه جای 0x36 استفاده می کنیم.

```
IN R0,0x16
```

مثال: دستور IN قابلیت کار کردن با اسامی ثباتهای I/O را دارد:

```
IN R0,PINB
```

- سایر تفاوتهای IN و LDS
  - دستور IN دو بایتی است ولی LDS چهار بایتی
  - دستور IN در یک سیکل ماشین اجرا می‌شود، LDS در دو سیکل
  - در دستور IN امکان استفاده از نام ثباتها وجود دارد
  - IN تنها برای دسترسی به I/O است، LDS تمام حافظه را پوشش می دهد
  - IN در تمام AVRها وجود دارد اما LDS این طور نیست

AVR assembly – instr. types

- Arithmetic and logic
- Branch, jump
- Data movement
- Bit manipulation, bit test

AVR assembly – instructions

Arithmetic and logic

a+b	ADD
a-b	SUB
a&b	AND
a|b	OR
a++	INC
a--	DEC
-a	NEG
a=0	CLR
…	…

Move

reg1=reg2	MOV
reg=17	LDI
reg=mem	LDS
reg=*mem	LD
mem=reg	STS
*mem=reg	ST
periperal	IN
peripheral	OUT
heap	PUSH
heap	POP
…	…

Bit op., others

a<<1	LSL	Logical Shift Left
a>>1	LSR,	
Ø C Not avail. in C	ROL, ROR	Rotate Left Through Carry
Status bits	SEI, CLI, CLZ...	Set Interrupt
No op.	NOP	
…	…	

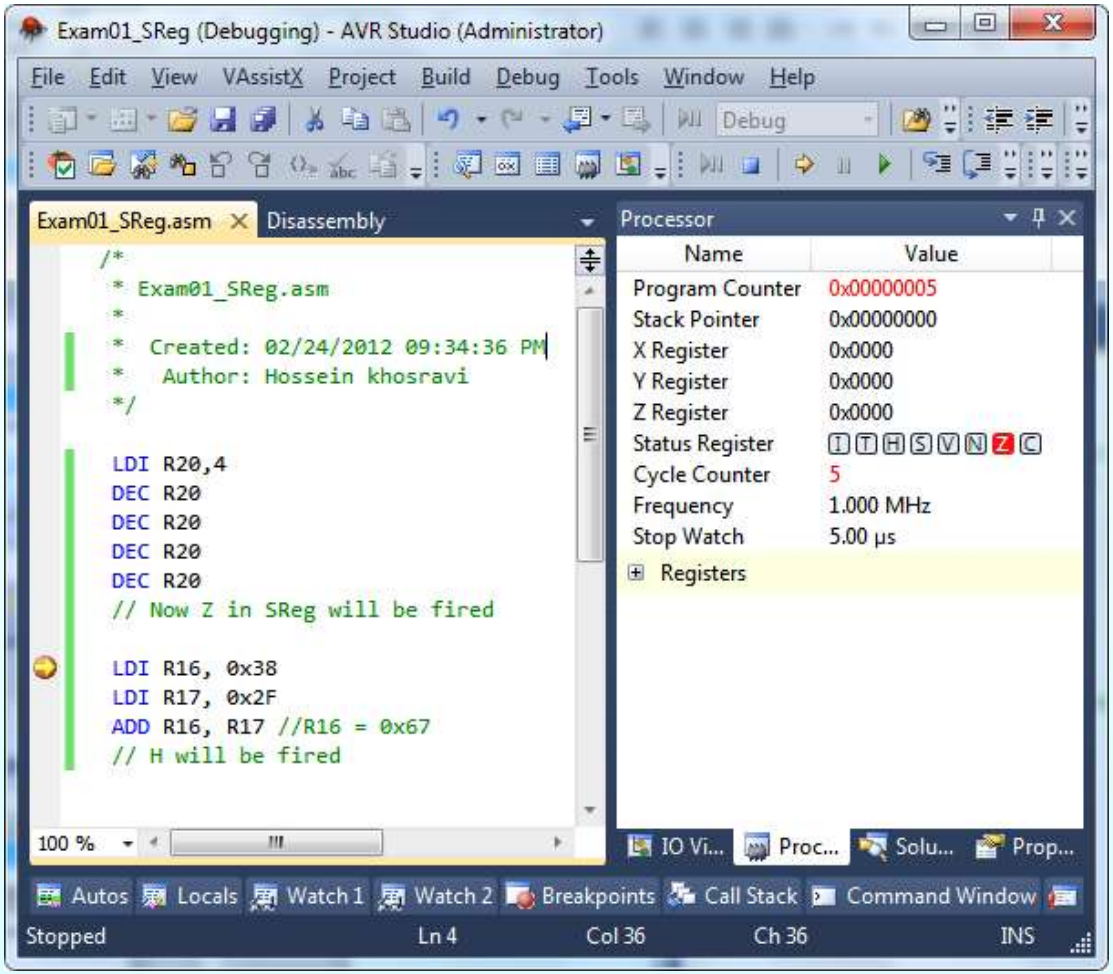
AVR assembly – special registers

- **Status Register (SREG)** – flags
- C: Carry Flag (from D7)
- Z: Zero Flag (after a Math)
- N: Negative Flag (in signed digits)
- V: Two's complement overflow indicator
- S: N ⊕ V, For signed tests
- H: Half Carry Flag (from D3 to D4)
- T: Transfer bit used by BLD and BST instructions
  - Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit.
- I: Global Interrupt Enable/Disable Flag

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

مثال

وضعیت پرچم های Z و H و C در طی اجرای برنامه؟



پس از اجرای خط پنجم پرچم Z یک می شود.

در پایان برنامه پرچم H یک می شود.

پرچم C در این برنامه همواره صفر است

هر دستوری روی SReg تاثیر نمی گذارد، برخی دستورات ریاضی و منطقی مثل ADD, INC, DEC, ROL, ROR, AND, OR, ... تاثیر دارند.

جدول ۲-۵ کتاب

AVR assembly – special registers

- Stack pointer
  - To store return address of subroutines, or save/restore variables (push, pop)
  - Grows from higher to lower addrress
  - 2 byte register
  - Stack stored in the data SRAM
  - FILO

```
ldi temp, LOW(RAMEND)
out SPL, temp
ldi temp, HIGH(RAMEND)
out SPH, temp
```

- Program Counter
  - Address of the actual instruction
  - During CALL or IT it is saved to the heap; RET/RETI loads from heap at the end of a subroutine/IT routine

نکاتی در مورد شمارنده برنامه و حافظه کد و داده

- در خانواده AVR هر خانه حافظه flash، ۲ بایت پهنا دارد
  - مثلا در ATmega32 ساختار حافظه 16kx16 است.
  - لذا ۱۴ خط آدرس دهی دارد.
  - از این رو ثبات شمارنده برنامه هم ۱۴ بیتی است.
- با اتصال برق، AVR از چه آدرسی شروع به کار می کند؟
  - آدرس 0x0000
- حافظه داده (SRAM) ۸ بیتی بوده و بایت به بایت آدرس پذیر است.

- با توجه به معماری هاروارد و مطالب فوق
  - در سمت داده ها، گذرگاه داده ۸ بیت و گذرگاه آدرس ۱۶ بیت است که قابلیت پوشش ۶۴ کیلوبایت RAM و I/O را دارد.
  - در سمت کد، گذرگاه داده ۱۶ بیت و گذرگاه آدرس ۱۴ بیت است.

ویژگیهای معماری RISC

- اندازه دستورات ثابت است
  - سرعت دیکد کردن دستورات افزایش می یابد
  - ماجرای بنایی که با آجرهای هم اندازه سر و کار دارد.
  - غالب دستورات AVR ۲ بایتی هستند. تعداد بسیار کمی ۴ بایتی اند.
  - در میکروی ۸۰۵۱ که CISC است، دستورات ۱، ۲، ۳ و چهاربایتی اند:
    - CLR C یک بایتی
    - ADD A, #25H ۲ بایتی
    - LJMP Lbl ۳ بایتی
    - CJNE A,B,Lbl ۴ بایتی
- تعداد زیاد ثبات
  - پردازنده های RISC حداقل ۳۲ ثبات همه منظوره دارند
- مجموعه دستورات محدود
  - ATmega حدود ۱۳۰ دستور دارد.
  - پیچیدگی کمتر سخت افزار و کدنویسی سخت تر اسمبلی
  - تمایل به استفاده از زبانهای سطح بالا

ویژگیهای معماری RISC

- حدود ۹۵٪ دستورات در یک کلاک اجرا می شوند
- معماری هاروارد
  - گذرگاههای مستقل برای دسترسی به کد و داده
  - ۲ گذرگاه (داده و آدرس) برای فضای داده و ۲ گذرگاه برای فضای کد
- فضای اشغالی کمتر نسبت به CISC
- معماری Load/Store
  - در میکروهای CISC داده‌های موجود در RAM می توانند دستکاری شوند مثلا در دستور ADD Mem, Reg داده از RAM خوانده شده با Reg جمع می شود و در نهایت در RAM قرار می گیرد. دسترسی به حافظه خارجی می تواند با تاخیر همراه شود و کل پروسه pipelining کند می شود.
  - در RISC عملیات تنها توسط ثباتها صورت می گیرد و از حافظه تنها برای Load/Store استفاده می شود

تمرین

- فصل ۱
  - ۱۵ سوال از ۳۰ سوال پایانی (زوج یا فرد)
- فصل ۲
  - ۲۵ سوال از ۸۰ سوال پایانی