ECED3204: Microprocessor
Part I--Introduction

Jason J. Gu
Department of Electrical and Computer Engineering
Dalhousie University

Outline

i. Computer
ii. Processor
iii. Embedded System
iv. Memory
v. Program Execution
VI. AVR
VII. AVR Memory
VIII. AVR CPU Register
IX. AVR Instruction
X. AVR Addressing Mode

I: Introduction of the Computer

- Computer terminology: hardware, software, Boolean algebra, bit, byte, word, double word
- Computer components: processor, input devices, output devices, and memory
- Categories: mainframe computers, minicomputers, microcomputers, and supercomputers

bit, byte, word, double word

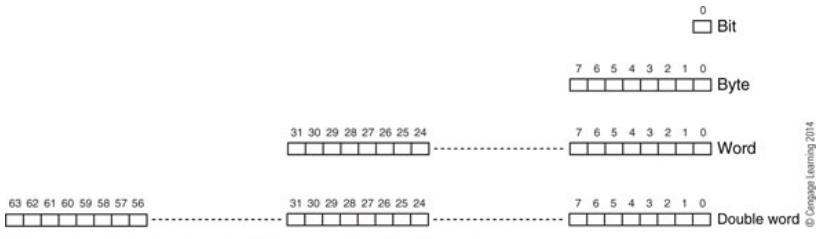
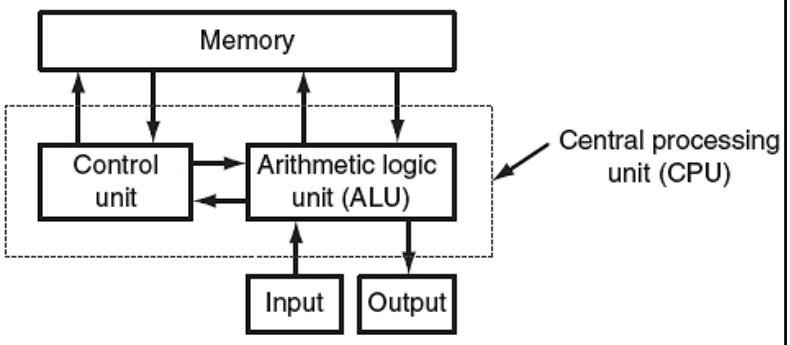


Figure 1.1 ■ Relative sizes of a bit, byte, word, and double word

von Neumann architecture



Proposed by John von Neumann in 1945

Computer Categories

- Mainframe computers
  - Original means huge size, now means very fast, high reliable, extensive I/O, Backward Compatible
- Minicomputers
  - Smaller than Mainframe, appeared in 1960s
- Microcomputers
  - Uses a microprocessor as its CPU
- Supercomputers
  - Super fast, using multiple processor

Computer Hardware Organization

- Components interconnected by buses
  - Processor: computational operations
  - Input devices: means of entering programs and data
  - Output devices: provide access to end results
  - Memory devices: provide the processor ready access to programs and data

II: The Processor

- The arithmetic logic unit (ALU)
  - performs arithmetic and logic operations
- Registers
  - high-speed memory circuits
- The control unit
  - reads and interprets program instructions
- The language issue
  - Computer executes machine instructions only
  - Low-level vs. high-level languages

An ALU Example

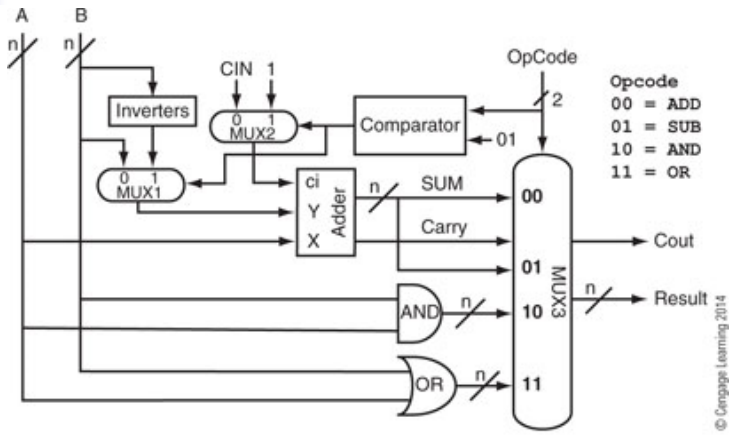


Figure 1.3 ■ An ALU that implements ADD, SUB, AND, and OR operations

The Microprocessor

- Single integrated circuit
- Advantages: small size, programmability, ease-of-use, and low cost
- Disadvantages: lacks on-chip memory; cannot drive I/O devices directly; lacks peripheral functions

The Microcontroller (MCU)

- Implemented on very large-scale integrated (VLSI) circuit
  - Incorporates the processor and following: Memory, timer, communication, A/D,D/A, I/O and so on.
- Typical applications
  - Controllers for displays, printers, keyboards, modems, charge-card, and home appliances
  - Control operations of factory engines and machines
  - Automobile control: Engine Controller

III: Embedded Systems

- Special-purpose computer system designed to perform a dedicated function
- Examples
  - digital watches, MP3 players, traffic lights, factory controllers, power plant control systems
- Can be optimized, and thus reduce size and cost
- Often mass-produced

IV: Memory

- Major memory technologies
  - Magnetic memory: hard disk still used
  - Optic memory
    - Compact disc (CD): CD-R, CD-RW
    - Digital video disc (DVD): DVD-R, DVD-RW, Blu-ray
  - Semiconductor memory
    - Used with embedded systems
    - Categorized by volatility and read-writability

Semiconductor Memory (cont'd.)

- Nonvolatile and volatile memory
  - Nonvolatile: information kept without power
  - Volatile: information lost without power
- Random access memory (RAM)
  - Volatile or nonvolatile, read-write memory
  - Types: dynamic RAM (DRAM), static RAM (SRAM), magnetoresistive RAM (MRAM), ferroelectric RAM (FRAM)

Semiconductor Memory (cont'd.)

- Read-only memory
  - Nonvolatile
  - Only allows data to be read (some exceptions)
  - Types: mask-programmed read-only memory (MROM), programmable read-only memory (PROM), erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), flash memory

Memory-System Operation

- Consists of one or multiple memory chips
- m x n organization
  - m: number of memory locations
  - n: number of bits in each location
- Each memory location: contents and address
- Memory access types: read and write

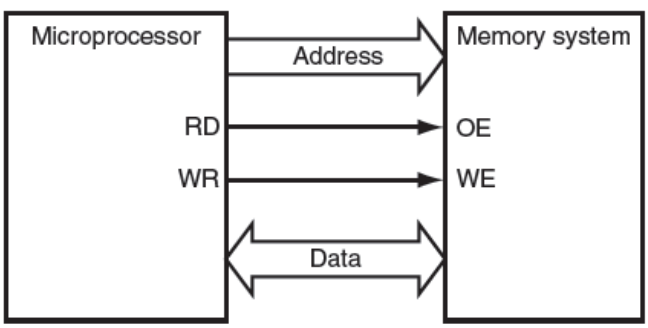


FIGURE 1.4 Block diagram of a simplified memory system

Note: RD read, WR write, OE output enable, WE write enable

Memory-System Operation (cont'd.)

- Read operation
  - Processor sends memory address of location to be accessed; applies logic 1 (high voltage) to RD signal and logic 0 to WR signal
  - Memory system decodes address; address contents sent to data bus; read by processor

Memory-System Operation (cont'd.)

- Write operation
  - Processor: data placed on data bus, address placed on address bus; logic 1 applied to WR signal, logic 0 to RD signal
  - Memory system: address decoded; value on data bus written to location

V: Program Execution

- Mainframe computers, minicomputers, personal computers
  - Power on: basic input/output system (BIOS) in nonvolatile memory performs system initialization
  - After initialization: processor loads other programs from secondary storage
- Other computers:
  - All programs in nonvolatile memory

Program Execution (cont'd.)

- The program counter circuit
  - Consists of flip-flops and other additional logic gates
  - Several types of flip-flops: D-type flip-flop most popular

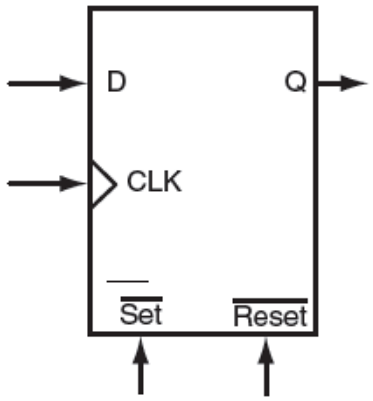


FIGURE 1.5 Block diagram of a D flip-flop with set and reset

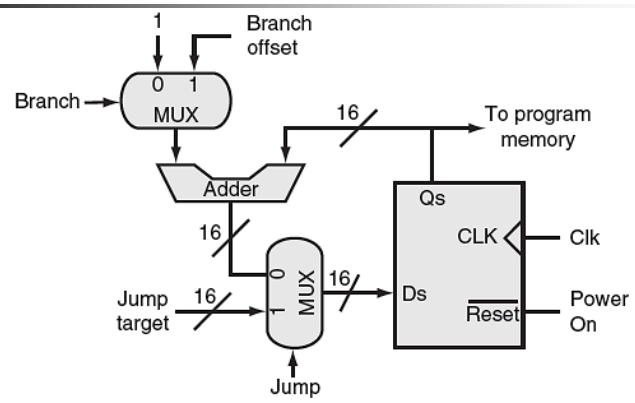


FIGURE 1.6 A simplified block diagram of the program counter (PC) of an 8-bit microcontroller

Program Execution (cont'd.)

- Starting program execution
  - One approach: force PC to a fixed value when power is turned on
  - Another approach: fetch program starting address from a fixed (known) memory location when power is turned on
  - Reset: identical to turning power on

Program Execution (cont'd.)

- Instruction execution process
  1. Read next instruction indicated by PC
  2. Decode the fetched instruction
  3. Increment the PC
  4. Read data the instruction requires from memory (or input device)
  5. Provide the necessary data to ALU or register

Instruction Execution Process (cont'd.)

6. If instruction requires ALU or specialized hardware to complete, processor instructs hardware to perform requested operation
7. Write result from ALU back to a memory location or to a register or perhaps an output device
8. Jump back to step 1

VI: Introduction to AVR Microcontroller

- AVR architecture provides
  - Program and data stored in separate physical memory address space
  - Ability to read data items from program memory using special instructions
- Four subgroups:
  - tinyAVR, 1-8kB Mem, 54-120 Instructions, 6-32pin, 20MHz
  - Mega AVR,4-256kB, 130-135 Instructions, 28-100pin, 20MHz
  - Xmega, 32-384kB, 142 Instruction,44-100pin, 32MHz
  - Application-specific AVR

VII: The AVR Memory Space

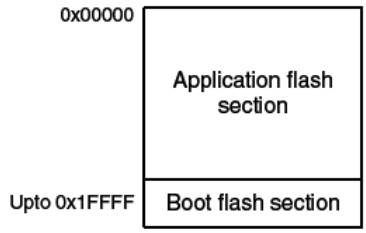


Figure 2.1 MEGA AVR program flash memory map

Program memory: m*16 matrix

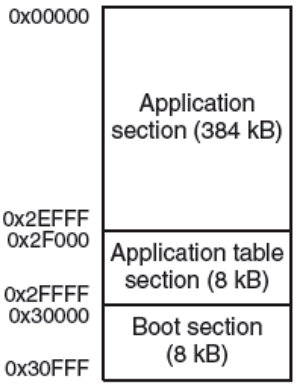


Figure 2.2 XMEGA384A1 program memory map

The AVR Memory Space (cont'd.)

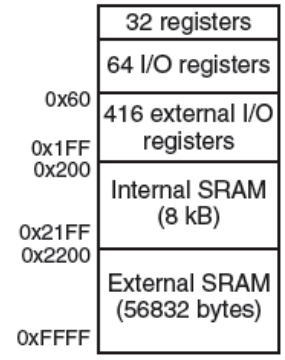


Figure 2.3 MEGA2560 data memory map, 64KB

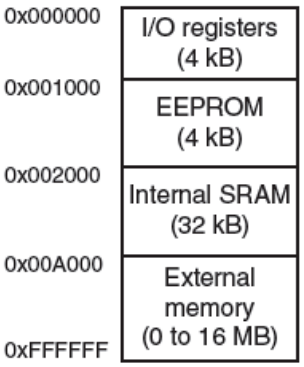


Figure 2.4 XMEGA384A1 data memory map, 16MB

VIII: The AVR CPU Register

	Addr.	
R0	0x00	
R1	0x01	
R2	0x02	
•••		
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
•••		
R26	0x1A	X-register low byte (XL)
R27	0x1B	X-register high byte (XH)
R28	0x1C	Y-register low byte (YL)
R29	0x1D	Y-register high byte (YH)
R30	0x1E	Z-register low byte (ZL)
R31	0x1F	Z-register high byte (ZH)

32-bit general purpose register with single clock cycle access time

Figure 2.5 ■ AVR CPU general purpose registers

The AVR CPU Register

- Thirty-two 8-bit general-purpose registers with single clock cycle access time
- X, Y, and Z registers: 16-bit address pointers for addressing data memory
- Z register can be used to access program memory
-

The AVR CPU Register

- RAMPX, RAMPY, and RAMPZ registers: concatenated with X, Y, and Z pointers to access data memory above 64 kB
- RAMPZ and Z pointer for program memory

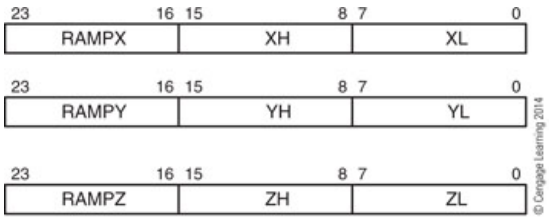


Figure 2.6 ■ The combined RAMPX + X, RAMPY + Y, and TAMPZ + Z registers

The AVR CPU Register (cont'd.)

- The extended indirect register (EIND): supports extended indirect subroutine call and indirect jump in devices with program memory larger than 128 kB

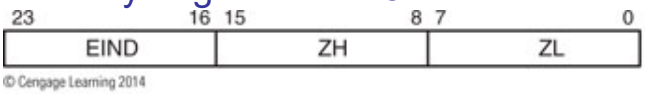


Figure 2.7 ■ The Z-pointer used by EICALL and EIJMP instructions

- The RAMPD register: supports direct addressing of the whole data memory space above 64 kB (XMega)

The AVR CPU Register (cont'd.)

- The status register (SREG): contains information about the result of the most recently executed arithmetic or logic instruction

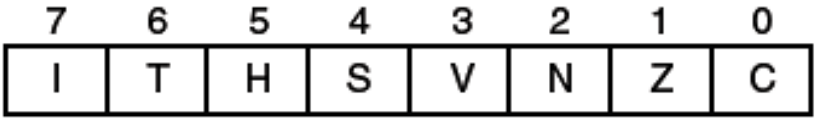


Figure 2.9 The SREG register

I: Global interrupt enable flag, chapter 9

T: Bit copy storage

H: Half carry flag

S: Sign bit

V: Two's complement overflow flag

N: Negative flag

Z: Zero flag

C: Carry flag

IX: The AVR Instruction Set

- Instruction parts: opcode and operand fields
- 142 instructions with 16-bit and 32-bit instructions
- Instruction categories: arithmetic, control flow, data transfer, bit field manipulation, logical, and miscellaneous
  - When an instruction has two operands, the first operand is used as a source and destination
  - sub r10,r11
- Summary in Appendix A of text

X: AVR Addressing Modes

- 1. Register direct mode: five bits used to specify register to be worked on
- Example:
  - sub r0,r1 ;r0←[r0]-[r1]
  - or r2,r3 ;r2←[r2]|[r3]
  - inc r4 ;r4←[r4]+1
- Note:
  - [ ] to refer to the content of a register or a mem location
  - mem(addr) refer to a data memory location
  - pmem(addr) refer to a program memory location

AVR Addressing Modes

- 2. I/O direct mode: six bits used to specify one of the first 64 I/O registers as an operand of an instruction
- Example:
  - in rd, A ; rd←I/O[A]
  - Out A,r1 ;I/O[A]←[r1]
- Note:
  - I/O registers in the extended I/O memory space must be accessed using load(LD,LDD/LDS) and (ST/STS/STD) instructions

AVR Addressing Modes

- 3. Direct data mode: 16-bit value used to specify the operand address
- Example:
  - lds r1,0x1010 ;r1←[0x1010]
  - sts 0x1011,r2 ;mem(0x1011) ←[r2]

AVR Addressing Modes (cont'd.)

- 4. Data indirect with displacement mode: data memory address formed by adding 6-bit field from the instruction and the Y or Z pointer
- example:
  - ldd r0,Z+10 ; r0←[[Z]+10]
  - std Z+20,r1 ;mem([Z]+20)←[r1]

AVR Addressing Modes (cont'd.)

- 5. Data indirect mode: contents of one of the pointers (X, Y, or Z) used to specify the address of the data memory location to access
- example:
  - ld r0,Z ; r0←[[Z]]
  - st Z,r1 ;mem([Z])←[r1]

AVR Addressing Modes (cont'd.)

- 6. Data indirect with pre-decrement: contents of the pointer X, Y, or Z are decremented by 1 and then used as an address to access data memory
- example:
  - ld r0,-Z ; Z←[Z]-1;r0←[[Z]]
  - st -Z,r1 ; Z←[Z]-1;mem([Z])←[r1]

AVR Addressing Modes (cont'd.)

- 7. Data indirect with post-increment: memory location pointed to by the pointer X, Y, or Z accessed and then pointer incremented by 1
- example:
  - ld r0,Z+ ; r0←[[Z]] ; Z←[Z]+1;
  - st Z+,r1 ; mem([Z])←[r1]; Z←[Z]+1;

AVR Addressing Modes (cont'd.)

- 8. Program memory constant addressing using the LPM, ELPM, and SPM instructions: use the Z pointer to specify program memory location to access
  - ELPM: RAMPZ register is used to extend the Z pointer
- example:
  - lpm r0,Z ; r0←[[Z]] ;
  - elpm r1,Z ; r1←[[RAMPZ:Z]];
  - spm Z+ ; (RAMPZ:Z) ←r1:r0;Z ←[Z]+2

AVR Addressing Modes (cont'd.)

- 9. Program memory with post-increment using the LPM Z+ and ELPM Z+ instructions
  - LPM Z+: uses Z pointer to specify program memory location to access; Z pointer incremented after the access
  - ELPM Z+: uses concatenation of RAMPZ and Z for program memory access; Z pointer incremented after the access
- example:
  - lpm r0,Z+ ; r0←[[Z]] ;Z←[Z]+1
  - elpm r1,Z+ ; r1←[[RAMPZ:Z]]; Z←[Z]+1

AVR Addressing Modes (cont'd.)

- 10. Direct addressing, JMP and CALL: 32-bit instructions
  - JMP: specifies the 22-bit address of the target instruction to jump to
  - CALL: specifies the 22-bit address of the subroutine to be called

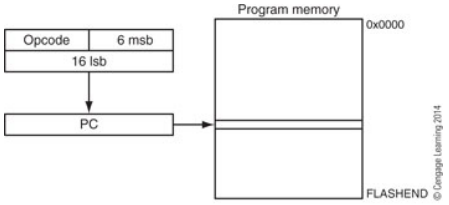


Figure 2.11 ■ Data program addressing, JMP and CALL

AVR Addressing Modes (cont'd.)

- 11. Indirect program addressing, IJMP and ICALL
  - IJMP: Z pointer to specify address of the target instruction to jump to
  - ICALL: Z pointer to specify address of the subroutine to be called

AVR Addressing Modes (cont'd.)

- 12. Relative program addressing, RJMP and RCALL
  - 12-bit value used to specify distance (from the instruction after RJMP) of the target instruction to jump to
  - RCALL: 12-bit value used to specify distance (from the instruction after RCALL) of the subroutine to call

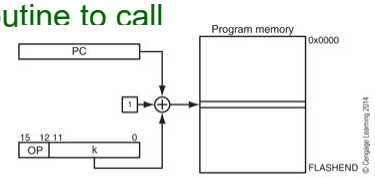


Figure 2.12 ■ Relative program addressing, RJMP and RCALL

A Sample of AVR Instructions

- Data movement, addition, and subtraction instructions examined
- AVR arithmetic and logic instructions require operand values copied from memory location to register before operation performed

A Sample of AVR Instructions

- Data transfer instructions supported by the AVR
  - Register to register: mov r0,r1
  - Constant loaded in a register: ldi rd,k
  - Data memory location to a register: lds r0, 0x100
- Examples:
  - Ldi r20,16 ; r20 ← 16
  - Ldd r20,z+5 ; r20 ←[[z]+5]
  - Lds r20,0x0100 ; r20 ←[0x0100]

Instruction	Description	Operation
lds Rd, K	Load direct from data memory	Rd ← [k]
ld Rd, X	Load indirect	Rd ← [[X]]
ld Rd, X+	Load indirect and post-increment	Rd ← [[X]]; X ← [X] + 1
ld Rd, −X	Load indirect and pre-decrement	X ← [X]−1; Rd ← [[X]]
ld Rd, Y	Load indirect	Rd ← [[Y]]
ld Rd, Y+	Load indirect and post-increment	Rd ← [[Y]]; Y ← [Y] + 1
ld Rd, −Y	Load indirect and pre-decrement	Y ← [Y]−1; Rd ← [[Y]]
ldd Rd, Y + q	Load indirect with displacement	Rd ← [[Y] + q]
ld Rd, Z	Load indirect	Rd ← [[Z]]
ld Rd, Z+	Load indirect and post-increment	Rd ← [[Z]]; Z ← [Z] + 1
ld Rd, −Z	Load indirect and pre-decrement	Z ← [Z]−1; Rd ← [[Z]]
ldd Rd, Z + q	Load indirect with displacement	Rd ← [[Z] + q]

Table 2.1 Instructions to transfer data from data memory to register

Data Transfer Instructions Supported by the AVR (cont'd.)

- Register to a data memory location
- Program memory location to a register
- Register to a program memory location
- I/O operation

Instruction	Description	Operation
sts K, Rr	Store direct to data memory	Mem (K) ← [Rr]
st X, Rr	Store indirect	Mem ([X]) ← [Rr]
st X+, Rr	Store indirect and post-increment	Mem ([X]) ← [Rr]; X ← [X] + 1
st −X, Rr	Store indirect and pre-decrement	X ← [X]−1; mem([X]) ← [Rr]
st Y, Rr	Store indirect	Mem([Y]) ← [Rr]
st Y+, Rr	Store indirect and post-increment	Mem([Y]) ← [Rr]; Y ← [Y] + 1
st −Y, Rr	Store indirect and pre-decrement	Y ← [Y]−1; mem([Y]) ← [Rr]
std Y + q, Rr	Store indirect with displacement	Mem([Y] + q) ← [Rr]
st Z, Rr	Store indirect	Mem([Z]) ← [Rr]
st Z+, Rr	Store indirect and post-increment	Mem([Z]) ← [Rr]; Z ← [Z] + 1
st −Z, Rr	Store indirect and pre-decrement	Z ← [Z] - 1; mem([Z]) ← [Rr]
std Z + q, Rr	Store indirect with displacement	Mem([Z] + q) ← [Rr]

Table 2.2 Instructions to store the contents of a register in data memory

Instruction	Description	Operation
lpm	Load program memory	r0 ← [[Z]]
lpm Rd, Z	Load program memory	Rd ← [[Z]]
lpm Rd, Z+	Load program memory and post increment	Rd ← [[Z]]; Z ← [Z] + 1
elpm	Extended load program memory	r0 ← [[RAMPD:Z]]
elpm Rd, Z	Extended load program memory	Rd ← [[RAMPD:Z]]
elpm Rd,Z+	Extended load program memory and post-increment	Rd ← [[RAMPD:Z]]; Z ← [Z] + 1
spm	Store program memory	pmem([RAMPD:Z]) ← r1:r0
spm Z+	Store program memory and post-increment by 2	pmem([RAMPD:Z]) ← R1:r0; Z ← [Z] + 2

Table 2.3 Instructions to transfer data between a register and the program memory

A Sample of AVR Instructions (cont'd.)

- Addition instruction

Instruction	Description	Operation
add Rd, Rr	Add without carry	Rd ← [Rd] + [Rr]
adc Rd, Rr	Add with carry	Rd ← [Rd] + [Rr] + C
adiw Rd, k	Add immediate to word	Rd + 1:Rd ← [Rd + 1]:[Rd] + k

Table 2.4 AVR addition instructions

Note:
Adiw rd,k; this instruction adds the constant k (0-63) to the register pair rd+1:rd and place the results in the register pair rd+1:rd , rd can only be 24,26,28,30

A Sample of AVR Instructions (cont'd.)

- Subtract instructions

Instruction	Description	Operation
sub Rd, Rr	Subtract without carry	Rd ← [Rd]−[Rr]
subi Rd, k	Subtract immediate	Rd ← [Rd] − k
sbc Rd, Rr	Subtract with carry	Rd ← [Rd]−[Rr] − C
sbci Rd, k	Subtract immediate with carry	Rd ← [Rd]−k − C
sbiw Rd, k	Subtract immediate from word	Rd + 1:Rd ← [Rd + 1]:[Rd] − k

Table 2.5 AVR subtract instructions

Note:
sbiw rd,k; this instruction subtract the constant k (0-63) from the register pair rd+1:rd and place the results in the register pair rd+1:rd , rd can only be 24,26,28,30