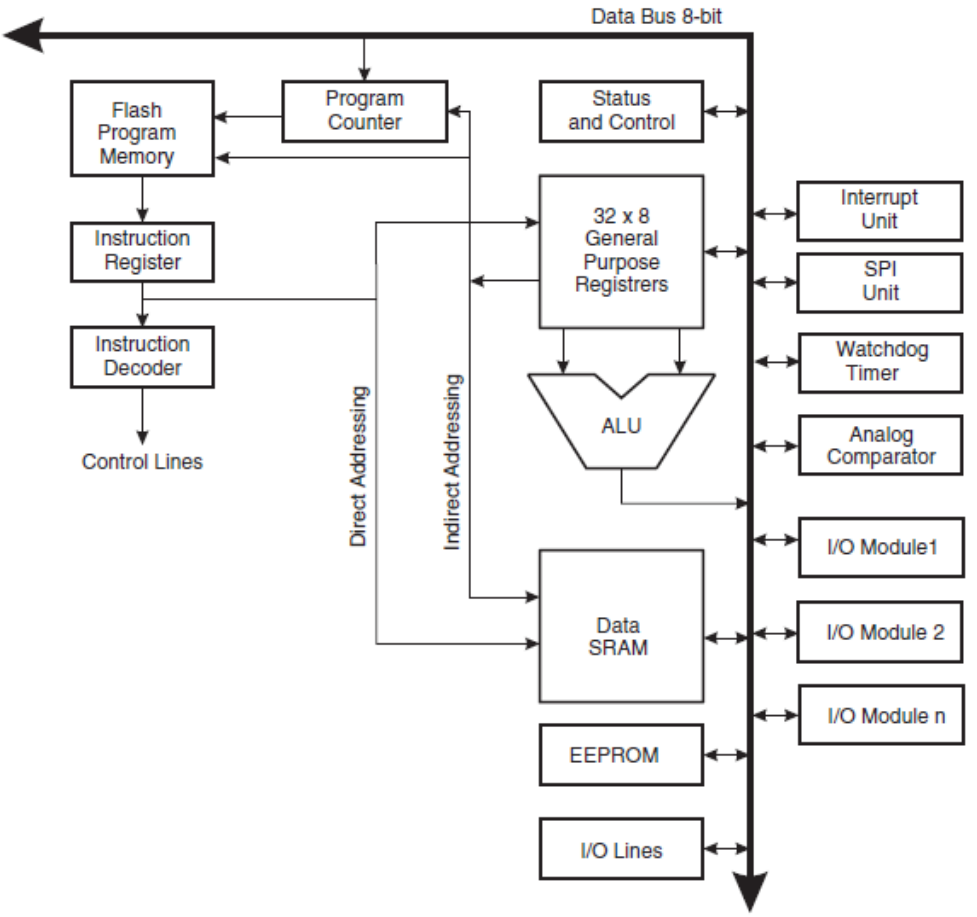
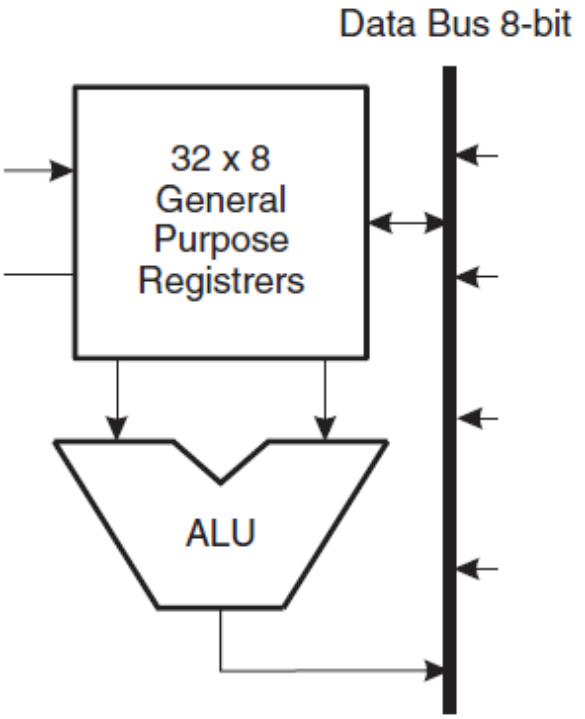
Atmega128 CPU Core (1)

- Overview
  - Harvard Architecture: 성능과 Parallelism 극대화 가능
  - 각각의 Instruction이 clock cycle마다 실행이됨.
    (instruction 하나 실행하면, 다음 instruction pre-fetch)
  - 32 X 8-bit general purpose Register 가지고 있음.
    그 중에서 X-, Y-, Z- regitster은 16-bit Data Space Addressing 가능
  - ALU: arithmetic / logic operation 지원
  - Status Register: operation 결과 저장
  - Flash Program Memory는 Boot Program Section과 Application Program Section으로 나뉘어져 있음.
  - Interrupt나 Subroutine Call 발생시 Program Counter Address가 Data SRAM에 Stack 형태로 저장 됨.
  - Interrupt는 Interrupt vector Table에 Interrupt vector로 존재하며 Interrupt마다 우선순위가 존재 함.

Figure 3. Block Diagram of the AVR Architecture



- ALU - Arithmetic Logical Unit
  - General Purpose Registers들과 Direct로 연결 되어 있음.
  - ALU Operation: 1) arithmetic (ADD, SUB ...) 2) logic (AND, OR ...) 3) bit-function (LSL : Logical Shift Left , LSR : Logical Shift Right ...) 더 자세한 Instruction은 DataSheet p)365 참고



- Status Register
  - 가장 최근에 실행된 Arithmetic instruction 결과 정보를 저장한다.
  - Status Register은 항상 ALU Operation이 끝난 후 업데이트 된다.
  - Interrupt 발생 시 혹은 Interrupt가 끝난 후 원래 상태로 돌아올 때 Status Register가 업데이트 되지 않으므로 이때는 Software로 handle 해야한다. (AVR status register: SREG)

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	

  - **Bit7 - I : Global Interrupt Enable**
    Interrupt 발생시키기 위해서는 SREG - Bit 7을 먼저 활성화 시켜야 한다.
    보통은 SREG = 0x80; 이나 sei(); 등으로 사용하여 global Interrupt Enable 시키기 위해 사용이 많이 된다.

```
int state = 0;

ISR(INT0_vect){
   state = !state;
}

int main(){
   DDRC = 0xFF;
   PORTC = 0xFF;
   SREG = 0x80; // sei();
```

```
   while(1){
      if(state) PORTC = 0xF0;
      else PORTC = 0x0F;
   }
}
```

Interrupt가 발생하면 다른 Interrupt에 의해 현재 Interrupt가 방해되지 않도록 Bit - I 가 Clear되고, return 될때 "RETI" instruction 에 의해 Bit - I가 다시 활성화 된다.

- [SEI instruction] : Global Interrupt Enable
- [CLI instruction] : Global Interrupt Disable

- Bit6 - T : Bit Copy Storage
  - [BST instruction] : (Bit STore) bit from register in Register file can be copied into Bit - T [register => Bit - T]
  - [BLD instruction] : (Bit LoaD) Bit - T can be copied into register in Register file [Bit - T => register]

- Bit5 - H : Half Carry Flag
  Half Carry Flag는 BCD(Binary Code Decimal) 연산 수행하기 위해 Half Carry를 활성화 하는 Flag이다.
  보통 User가 건드릴 일은 없겠지만, AVR에서 Decimal Arithmetic Instruction을 수행할 때 해당 Instruction이 BCD 연산을 이용한다면 Half-Carry Flag를 사용할 것이다.

  - Intel 80386 Decimal Arithmetic Instruction
    잠시 BCD 연산을 이용한 DAA 연산을 설명하기 위해 인텔 80386 Instruction을 들고 왔다.
    - 1) Packed BCD Instruction: DAA(Decimal after Addition), DAS(Decimal after Subtraction)
    - 2) Unpacked BCD Instruction: AAA(ASCII after Addition) , AAS(ASCII after Subtraction) , AAM(Multiplication), AAD(Division)
    - DAA, AAA 연산 차이

      (1) DAA 연산

```
IF ((AL AND 0FH) > 9) OR (AF = 1) // ((AL && 0x0F) > 9 || AF == 1)
THEN
   AL := AL + 6;
   AF := 1;
ELSE
```

```
   AF := 0;
FI;
IF (AL > 9FH) OR (CF = 1)         // (CF == 1)
THEN
   AL := AL + 60H;
   CF := 1;
ELSE CF := 0;
FI;
```

(내용 추가 예정)

(2) AAA 연산

```
IF ((AL AND 0FH) > 9) OR (AF = 1) // ((AL && 0x0F) > 9 || AF == 1)
THEN
   AL := (AL + 6) AND 0FH;
   AH := AH + 1;
   AF := 1;
   CF := 1;
ELSE
   CF := 0;
   AF := 0;
FI;
```

(내용 추가 예정)

- (해당 Flag를 internal flag 대신 user-visible flag로 정의한 이유: Interrupt에 의해서 DAA 연산이 제대로 수행되지 못할 수도 있기 때문에 user-visible flag로 정의 했다고 추측이 된다.)
  https://retrocomputing.stackexchange.com/questions/4693/why-does-the-z80-have-a-half-carry-bit/6392

- Bit4 - S : Sign Bit, S = N ⊕ V
  bit - N 과 bit - V 의 exclusive OR 값

- Bit3 - V : Two's Complement Overflow Flag
  2's Complement 연산에서 연산 값이 오버플로우 발생했을 때 활성화 됨.

- Bit2 - N : Negative Flag
  연산 값이 음수일 때 활성화 됨.

- Bit1 - Z : Zero Flag
  연산 값이 0일 때 활성화 됨.

- Bit0 - C : Carry Flag
  - ADC : (Rd <-- Rd + Rr + C)
  - SBC : (Rd <-- Rd + Rr - C)