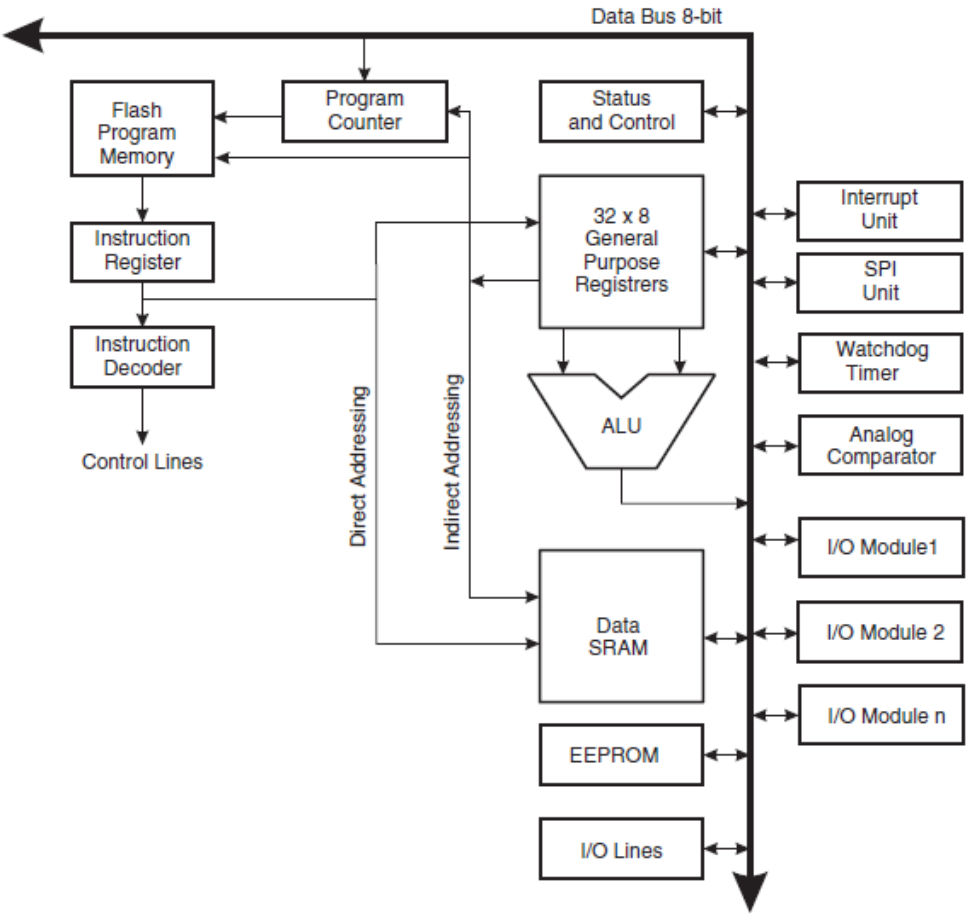
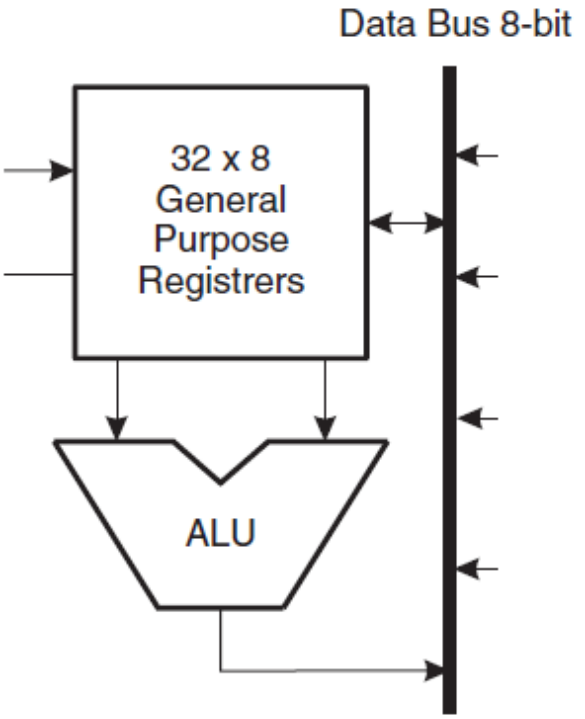
Atmega128 CPU Core (2)

- 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 Interrupt 마다 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)          // (AL > 0x9F) || (CF == 1)
THEN
   AL := AL + 60H;                 // AL += AL + 0x60;
   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)

- General Purpose Register File

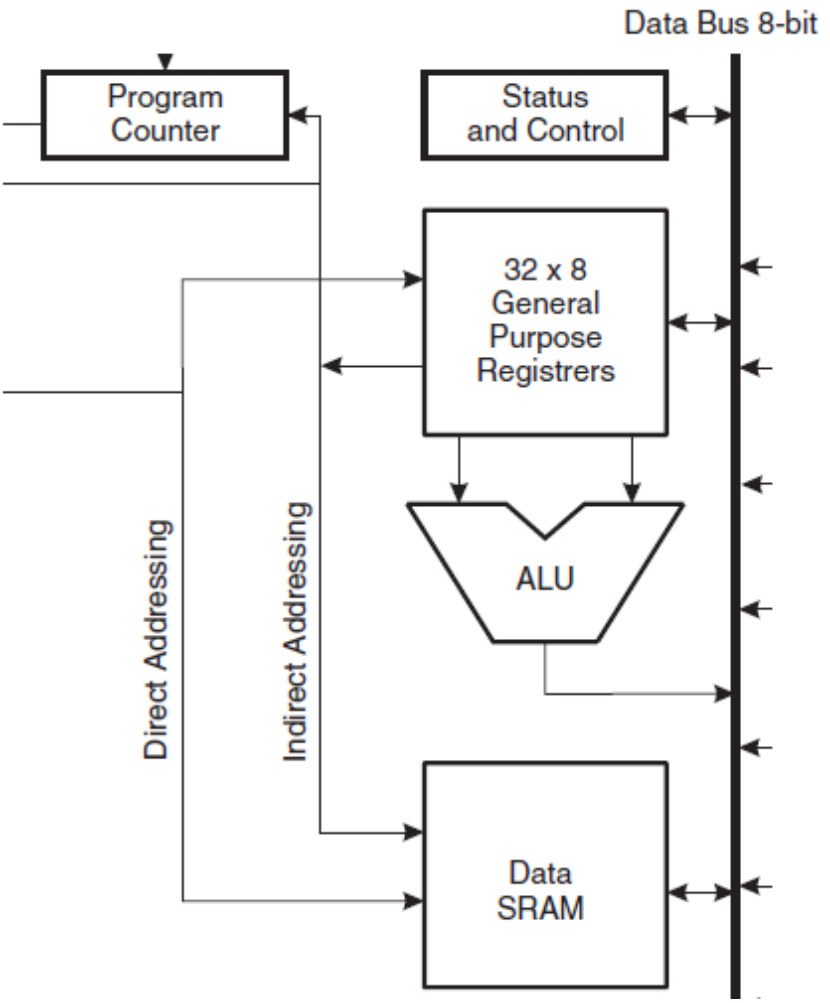


Figure 4. AVR CPU General Purpose Working Registers

	7 0	Addr.	
	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

- input / output Scheme (I/O 형태)
  - One 8-bit output operand and one 8-bit result input
  - Two 8-bit output operands and one 8-bit result input
  - Two 8-bit output operands and one 16-bit result input
  - One 16-bit output operand and one 16-bit result input

- X-, Y-, Z- registers : 16-bit address pointers for indirect addressing
  ex) (Load Program Memory 기능을 하는 LPM) : [LPM Rd Z] // Rd <= (Z)

- Stack Pointer : temporary data 저장하는 데 이용, Interrupt나 Subroutine 종료 이후 address return해서 받아옴.

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

일반적인 Stack과 달리 Stack 커질때 마다 Higher Memory Location에서 Lower Memory Location으로 이동함.
다음 표와 같이 PUSH 했을 경우, Stack Pointer가 줄어듬을 알 수 있음.

Syntax	Program Counter	Stack
PUSH Rr	PC <= PC +1	SP <= SP - 1
POP Rd	PC <= PC + 1	SP <= SP + 1

  - Subroutine 이나 Interrupt 발생 되기 전에 SRAM stack area에 Stack Pointer 가 지정 되어야 하며 SRAM 영역의 0x60 번지 이상으로 지정이 된다.
  - AVR Stack Pointer은 I/O Space에서 8bit register 2개 (SPL, SPH) 로 나눠어져 있으며 주로 SPL 영역이 자주 쓰인다.

- RAM Page Z Select Register - RAMPZ

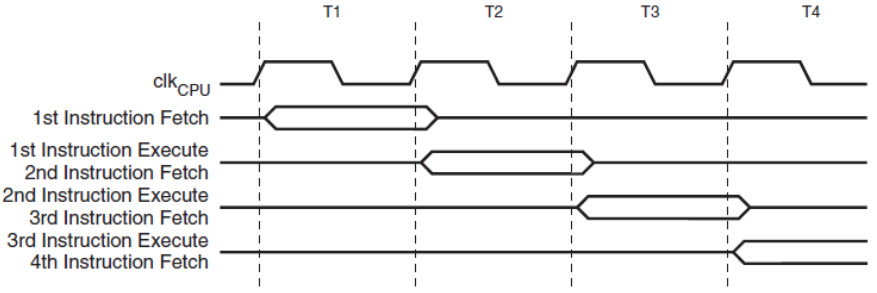
Bit	7	6	5	4	3	2	1	0	
	–	—	–	–	–	–	–	RAMPZ0	RAMPZ
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7..1 : Reserved Bits
  0로 지정 되어있으며, 다음 세대 device 호환성을 위해 남겨 짐.

- Bit 0 - RAMPZ0 : Extended RAM Page Z-pointer
  Z-pointer을 통해서 64K RAM Page 접속 가능하다.
  - RAMPZ0 = 0: (ELPM / SPM) 통해서 Program Memory Address [0x0000 - 0x7FFF : Lower 64Kbytes] 접근 가능하다.
  - RAMPZ0 = 1: (ELPM / SPM) 통해서 Program Memory Address [0x8000 - 0xFFFF : Higher 64Kbytes] 접근 가능하다.

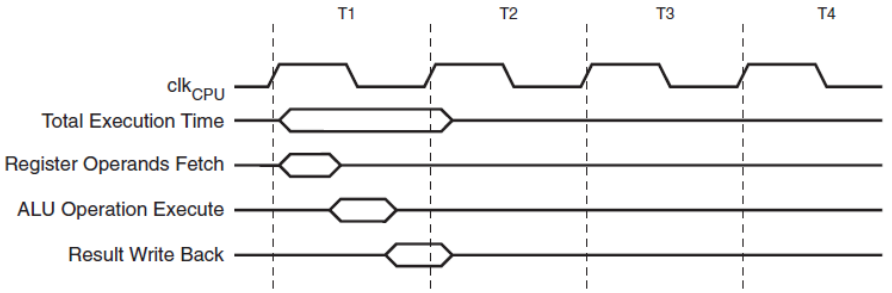
- Instruction Execution Timing
  - AVR CPU는 외부 클럭 소스로 생성된 clk_{cpu} 를 클럭신호로 가진다.

Figure 6. The Parallel Instruction Fetches and Instruction Executions



  - 다음 그림은 병렬 Instruction Fetch / Execution 과정이며, pipeline concept 의 기본이 되는 내용이다.

Figure 7. Single Cycle ALU Operation



  - 다음은 Register file 내의 internal time concept 인데, "(Register Fetch) - (ALU Execute) - (result Store Back)" 과정으로 이루어져 있다.

- Reset and Interrupt Handling
  - Interrupt를 수행하기 위해서는 Global Interrupt 핀을 활성화를 해야하며 지정된 Interrupt 혹은 Reset vector에서 enable bit를 활성화 시켜야 한다.
  - Program Memory Space 내에서 Reset과 Interrupt vector은 가장 낮은 Address로 지정되어 있음. (우선 순위가 높다는 의미인거 같다)
  - RESET이 가장 높은 우선 순위

- INT0(외부 인터럽트 0)가 2순위
- MCU Control Register(MCUSR)의 IVSEL bit를 설정함으로써 interrupt vector을 부팅 Flash section할때 실행시킬 수 있다.
- Reset vector은 BOOTRST을 통해 Boot Flash section 시작시 실행시킬 수 있다.

- Interrupt 발생 시, Global Interrupt I-bit과 다른 Interrupt들이 해제된다. (Nested Interrupt 실행시키기 위해서는 user S/W 상에서 Interrupt 발생시키면 된다.)
- Interrupt의 종류:
  - 1) Interrupt Flag에 의한 Event :
    Interrupt Handling Routine 수행을 위해 Program Counter에 Interrupt Vector로 만들어져 있으며, Interrupt 수행시 하드웨어 내에서
    해당 Interrupt에 대응하는 Interrupt Flag를 clear 시킨다.
    Global Interrupt Enable Bit가 Clear 된 상태에서 Interrupt Condition이 하나 이상 발생할 경우, 대응되는 Corresponding Interrupt Flag가 Set 되며
    Global Interrupt Enable Bit가 Set 되었을 때 우선순위에 따라 Interrupt가 수행 됨.

  - 2) Interrupt Condition이 지속되는 상태 :
    Interrupt Condition이 사라지지 않는 이상 Interrupt가 발생되지 않음.
- AVR은 인터럽트가 종료될 때 무조건 Main 프로그램으로 돌아와 다음 Interrupt가 발생되기 전까지 하나 이상의 Instruction을 수행함.
- Interrupt Routine에 들어가거나 나올 때, Status Registor은 자동으로 변하지 않기 때문에 software상에서 제어해야 한다.
- CLI Instruction : 모든 Interrupt가 해제됨.
- SEI Instruction : Global Interrupt 허용

- Interrupt Response Time
  - AVR에서 수행되는 Interrupt들은 **최소 4 Clock** 동안 수행된다. 그 이후에는 다시 Interrupt Handling Routine (정상 상태)이 수행된다.

  - [1. Interrupt Handling Routine -> Interrupt Routine]
    - Interrupt가 수행되는 4 Clock 내에서 Program Counter는 Stack에 Push 된다.
    - Interrupt Routine으로 상태를 바꾸는 데 **3 Clock** Cycle이 소요된다.
  - [2. Interrupt Routine -> Interrupt Handling Routine]

- Interrupt가 수행되는 4 Clock 내에서 Program Counter는 Stack에서 Pop back 된다.
- Interrupt Handling Routine으로 돌아오는 데 **4 Clock** Cycle이 소요된다.