

# AVR 프로그래밍

2강

정보컴퓨터공학과 권혁동

<https://youtu.be/Q9B5ypVY8oM>

# Contents

기초 어셈블리 구현

코드 분석



CryptoCraft LAB

# 기초 어셈블리 구현

- 지난 강의에서 구현한 코드를 어셈블리로 이식
  - 목표: 8-bit 덧셈기 구현
- 인라인 어셈블리: C코드 사이에 삽입되는 어셈블리
- 어셈블리: **.s(.S) 파일로 구성되는 어셈블리**

```
int main(void)
{
    int a = 1;
    int b = 2;
    int c;

    c = a + b;
}
```

als	
ame	Value
a	0x0001
b	0x0002
c	0x0003

# 기초 어셈블리 구현

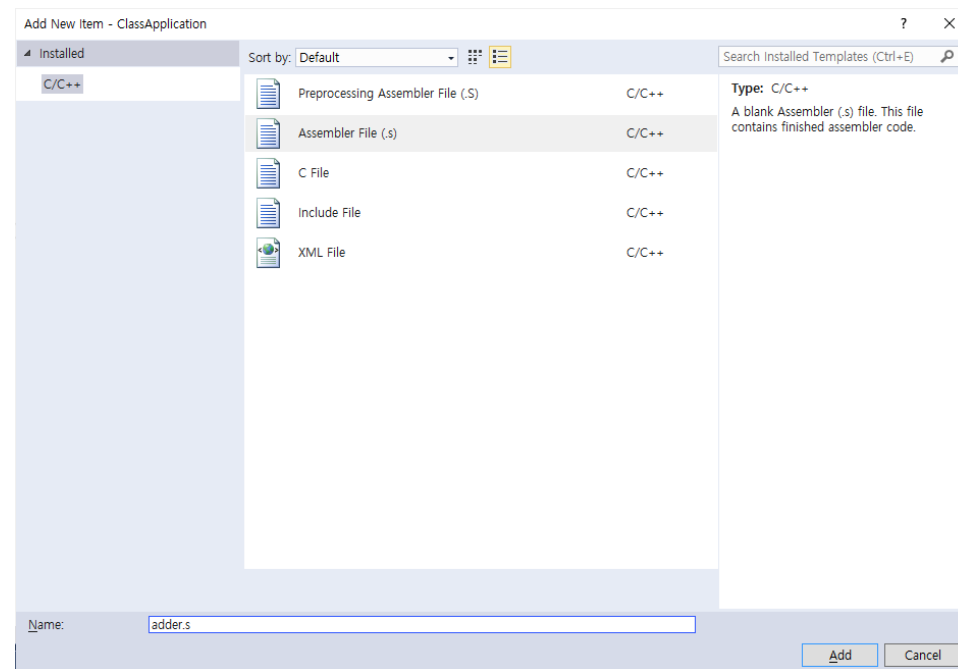
- 프로그램의 기본 내용을 다음과 같이 수정
  - 함수는 프로토타입으로만 작성
  - c파일 외부에 함수가 정의되므로 **extern 키워드** 사용
  - 모든 **매개변수는 포인터 형식**으로 입력
- 어셈블리 파일을 새로 생성
  - 파일 이름은 자유

```
extern void adder(char *x, char *y, char *z);
```

```
int main(void)
{
    char a = 1;
    char b = 2;
    char c;

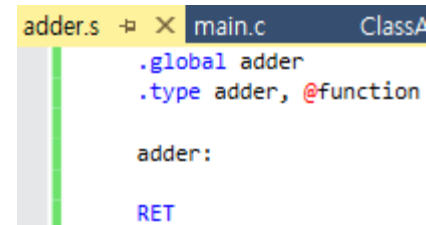
    adder(&a, &b, &c);

    //c = a + b;
}
```



# 기초 어셈블리 구현

- 생성한 어셈블리 파일에 다음과 같은 코드를 작성
  - `.global adder`  
`.type adder, @function`  
`adder:`  
`RET`
  - ‘RET’는 대소문자 구분 없음
  - ‘adder’은 함수 이름이므로, 다른 이름을 사용 가능



```
adder.s  x main.c  ClassA
.global adder
.type adder, @function

adder:

RET
```

# 기초 어셈블리 구현

- 사용할 명령어 모음

명령어	동작
LD	메모리에서 레지스터로 로드
ST	레지스터에서 메모리로 저장
MOVW	레지스터 2개(워드) 단위로 이동(복사)
ADD	레지스터 덧셈
RET	함수 반환

# 기초 어셈블리 구현

- 함수의 내용을 다음과 같이 정의
  - MOVW R26, R24
  - LD R18, X
  - MOVW R26, R22
  - LD R19, X
  - MOVW R30, R20
  - ADD R18, R19
  - ST Z, R18
  - RET
- 8-bit 덧셈기 구현 완료

```
adder.s  X main.c  ClassA
.global adder
.type adder, @function

adder:

    MOVW R26, R24
    LD R18, X
    MOVW R26, R22
    LD R19, X
    MOVW R30, R20

    ADD R18, R19

    ST Z, R18

    RET
```

# 기초 어셈블리 구현

- 디버깅으로 덧셈 구현 확인 가능

```
int main(void)
{
    char a = 1;
    char b = 2;
    char c = 0;

    adder(&a, &b, &c);

    //c = a + b;
}
```

```
int main(void)
{
    char a = 1;
    char b = 2;
    char c = 0;

    adder(&a, &b, &c);

    //c = a + b;
}
```

Variable	Value	Variable	Value
a	0x01	a	0x01
b	0x02	b	0x02
c	0x00	c	0x03



# 코드 분석

- MOVW Rd, Rr
- ADD Rd, Rr
- LD Rd, X
- ST Z, Rr
  - Rd: 목적지 레지스터
  - Rr: 출발 레지스터
  - X, Z: 포인터 레지스터

- **즉, MOVW R26, R24는  
R24, R25의 내용을 R26, R27로 이동(복사)**

```
• MOVW R26, R24
  LD R18, X
  MOVW R26, R22
  LD R19, X
  MOVW R30, R20
  ADD R18, R19
  ST Z, R18
  RET
```

# 코드 분석

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- R1: ZERO, 종료 시에는 항상 0으로 유지
- R2~R17, R28, R29: Callee Saved, 사용 이전의 값 보존이 필요함
- R26, R27: X pointer
- R28, R29: Y pointer
- R30, R31: Z pointer

# 코드 분석

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- 매개 변수는 (R24, R25), (R22, R23), (R20, R21) ... 순서로 입력
  - 변수의 주소 값이 16-bit 형태로 저장되어 있음
- **포인터로 활용하기 위해서는 X, Y, Z를 통해야 함**
- 따라서 MOVW 명령어를 통해 X, Y, Z 포인터로 사용가능한 레지스터로 값을 이동

# 코드 분석

- R20 ~ R25에 매개변수 주소 값이 저장
- 이 상태에서는 값을 불러올 수 없으므로, 각각을 X 또는 Z로 이동
  - Y는 callee saved 이므로 사용하지 않음
- LD를 통해서 메모리에 위치한 값을 가져올 수 있음

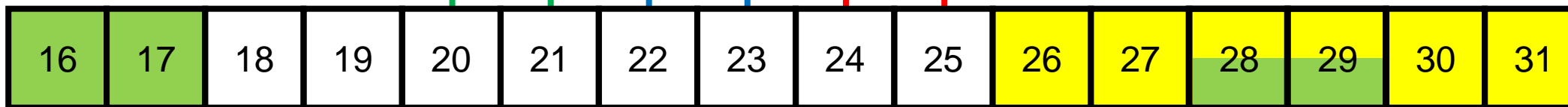
```
extern void adder(char *x, char *y, char *z);
```

```
int main(void)
{
    char a = 1;
    char b = 2;
    char c;

    adder(&a, &b, &c);

    //c = a + b;
}
```

c의 주소      b의 주소      a의 주소



```
adder.s  X main.c  Class
.global adder
.type adder, @function

adder:
    MOVW R26, R24
    LD R18, X
    MOVW R26, R22
    LD R19, X
    MOVW R30, R20

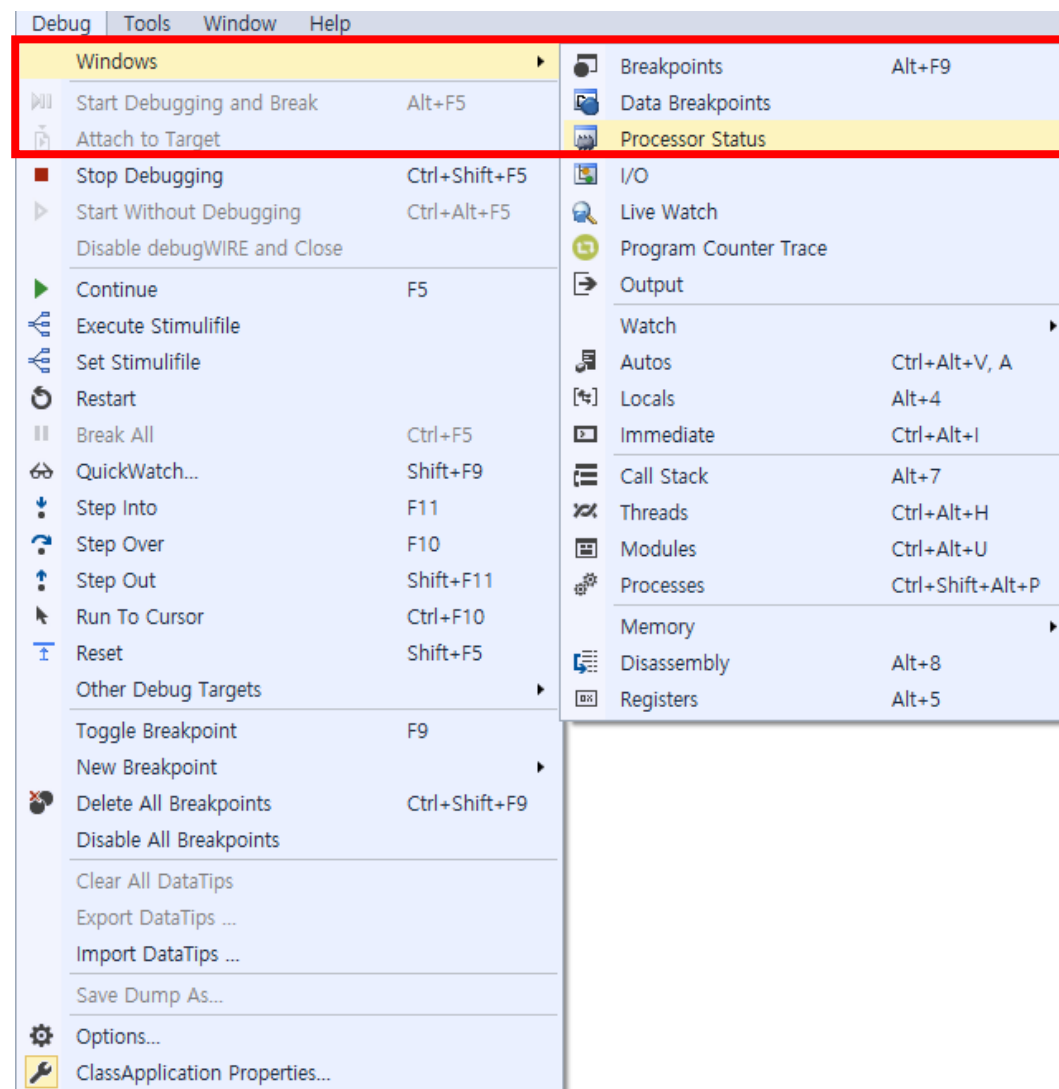
    ADD R18, R19

    ST Z, R18

    RET
```

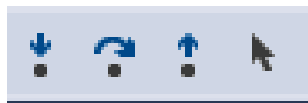
# 코드 분석

- 디버깅 중에 레지스터, 클록 사이클 등 값 추적을 위해 'Processor Status'를 활성화
- Debug → Windows → Processor Status



# 코드 분석

- 어셈블리 코드에 중단점을 생성하고 상태를 확인
  - R20 ~ R25까지 주소 값을 확인 가능
  - R18, R19는 R20, R21과 같은 값
  - 컴파일러가 자체적으로 생성한 코드로 인한 결과



## 디버깅 도구

- Step Into: 명령어 하나 동작, 함수나 점프가 있다면 내부로 진입
- Step Over: 명령어 하나 동작
- Step Out: 다음 중단점까지 건너 뛰기
- Run To Cursor: 커서가 있는 곳으로 건너 뛰기

Processor Status	
Name	Value
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0xFB
R19	0x10
R20	0xFB
R21	0x10
R22	0xFA
R23	0x10
R24	0xF9
R25	0x10
R26	0x00
R27	0x00
R28	0xF8
R29	0x10
R30	0x00
R31	0x00

# 코드 분석

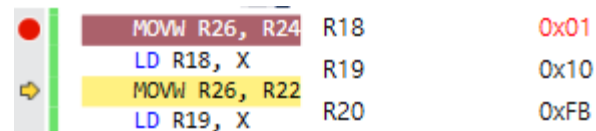
디버깅 도구 활용 왼쪽부터



- Step Into: 명령어 하나 동작,  
함수나 점프가 있다면 내부로 진입
- Step Over: 명령어 하나 동작,  
함수나 점프가 있다면 해당 내용을 다 수행한 것으로 처리
- Step Out: 다음 중단점까지 건너 뛰기
- Run To Cursor: 커서가 있는 곳으로 건너 뛰기

# 코드 분석

- Step Over를 사용하여 한 줄마다 코드 동작을 확인
- R18 레지스터에 변수 a의 값이 저장된 것을 확인 가능



MOVW R26, R24	R18	0x01
LD R18, X	R19	0x10
MOVW R26, R22	R20	0xFB
LD R19, X		



Q & A

