<MiniC – Ucode 예>

C 코드

```
const int max=500;
void main(){
	int i, j, k;
	int rem, sum;

	i = 2;
	while(i <= max) {
		sum = 0;
		k = i/2;
		j = i;
		while ( j <= k) {
			rem = i%j;
			if (rem == 0) {
				sum +=j;
			++j;
		}
		if  (i == sum) write(i);
		++ i;
	}
}
```

U 코드

```
main    proc    5   2   2
        sym     2   1   1           // i       2: base   1: offset   1:원소갯수
        sym     2   2   1           // j
        sym     2   3   1           // k
        sym     2   4   1           //rem
        sym     2   5   1           //sum
        ldc     2                   /// i = 2
        str     2   1               ///.
$$0     nop
        lod     2   1               /// i <= 500 .. const 변수는 그냥 값을 쓰기로 한다.
        ldc     500                 ///
        le                          ///.
        fjp   $$1                   /// 거짓이면 $$1으로 jump
        ldc     0                   /// sum = 0
        str     2   5               ///.
        lod     2   1               /// k = i / 2
        ldc     2                   ///
        div                         ///
        str     2   3               ///.
        ldc     1                   /// j   =1;
```

```
        str     2   2               /// .
$$2     nop
        lod     2   2               /// j <= k
        lod     2   3               ///
        le                          ///.
        fjp   $$3                   // 거짓이면 $$3으로
        lod     2   1               /// rem = i % j
        lod     2   2               ///
        mod                         ///
        str     2   4               ///.
        lod     2   4               /// rem == 0
        ldc     0                   ///
        eq                          ///.
        fjp    $$4                  // 거짓이면 $$4로
        lod     2   5               /// sum = sum +j
        lod     2   2               ///
        add                         ///
        str     2   5               ///.
$$4     nop
        lod     2   2               /// j++
        inc                         ///
        str     2   2               ///.
        ujp   $$2                   /// goto $$2
$$3     nop
        lod     2   1               /// i == sum
        lod     2   5               ///
        eq                          /// .
        fjp $$5                     // 거짓이면 $$5
        ldp                         /// write   호출    ... load parameter
        lod     2  1                /// i를 인자로 함
        call write                  /// .
$$5     nop
        lod     2   1               /// i++
        inc                         ///
        str     2   1               ///.
        ujp   $$0                   /// $$0으로 goto
$$1     nop
        ret                     /// 메인 끝
        end
```

```
bgn     0               // 프로그램 시작, 외부변수 크기는 0
ldp                     /// 메인호출
call main               /// .
end
```

참고

1. read, write, lf 함수 호출이 가능하다.

```
ldp
lda     b o     //base 와 offest
call    read

ldp
lod     b o     // base 와 offset
call    write

call    lf      // 라인 피드
```

2. 배열 변수가 들어 있을 때 처리

```
void main () {
        int list[100];
        int element;

        list[5] =10;
```

➔

```
main    proc    101     2       2
                // 101: 지역변수크기+매개변수크기, 2:블록번호 2: 렉시컬레벨
sym     2       1       100     /* list */
sym     2       101     1       /* element */

ldc     5
lda     2       1
add                             // list[5]의 주소값이 stack에 저장됨
ldc     10                      // 10을
sti                             // address로 store
```

기타

- 심벌테이블은 lookup과 insert가 되면 되므로 자료구조는 아무거나 편한 것으로 한다. (성능에 구애 받지 말고 시작할 것.)
- 기타 오세만저 컴파일러입문 정익사 9, 10장 참고
- C-like 하게 표현한 명령어 의미
    - notop : stack[top] = not stack[top]
    - neg : stack[top] = -stack[top]
    - inc : ++stack[top]
    - dec : -- stack[top]
    - dup: stack[top+1] = stack[top]; top = top+1
    - add : stack[top-1] = stack[top-1]+stack[top]; top = top – 1
    - gt : stack[top-1] = stack[top-1]>stack[top]; top = top – 1
    - swp : temp = stack[top-1]; stack[top-1] = stack[top]; stack[top] = temp