과제: TM 어셈블리 언어 및 실행 환경 구조 이해하기

1. 코드 이해 및 실습 (제공되는 내용)
- add.tm : 여기서 변수 a, b, c 는 스택이 아닌 dMem[] 의 아래 부분에 할당됨에 유의할 것
- fact.tm: 여기서는 startup 없이 main() 위주로 코딩 됨에 유의할 것
- funcall.tm (설명 파일: 4-cm2tm-ex.pdf) : startup이 포함된 일반적 형태의 실행 파일 구조
- stack.tm : 일반적 형태의 startup이 아님. 어떤 한 함수의 호출 및 로컬 변수의 구조만 표현.
- 1-startup.tm : startup에서 main함수를 호출하는 실행 파일 구조. 정작 main()은 아무 일도 안함.
- 2-localVar.tm : main()에서 주어진 두 변수의 값을 출력하는 기능을 함 (startup이 포함됨)
- 3번, 4번은 아래의 내용을 이용하여 실습

2, 제출 : 5~10번 문제에 대한 TM 코드와 수행 결과를 화면 캡쳐하고 pdf 파일로 만들어 제출

실습: 아래 예제 테이블의 왼쪽은 CM (C Minus) 언어로 작성된 코드로서, CM은 C 언어와 유사하다. 총 10개의 CM 코드에 해당되는 TM 어셈블리언어를 작성하고, 이것을 tm 어셈블러를 이용하여 그 결과를 확인하라. 처음 4개에 대해서는 답이 주어졌으므로, 이 내용을 잘 이해하고 실습하면서 그 결과를 확인하라.

1. C Startup (출력되는 값 없음)

void main(void)
{
}
	// ====================
	// c startup
	// ====================
	0: ld gp, 0(0)
	1: st 0, 0(0)
	2: lda fp, -0(gp)
	3: lda sp, -0(gp)
	4: push fp
	5: lda 0, 2(pc) // return address (8)
	6: push 0 // push the address
	7: ldc pc, 9 // jump to main
	8: halt
	// ====================
	// main()
	// ====================
	9: lda sp, -0(sp) // zero local variables
	10: ldc 27, 0 // return value = 0
	11: lda sp, 0(fp) // adjust sp
	12: ld fp, 0(fp) // restore old fp
	13: ld pc, -1(sp) // return
	// ====================

2. Local Variables 사용시

```
void main(void)
{
  int l1;
  int l2;

  l1 = 0;
  l2 = 1;
  output l1;
  output l2;
}
```

```
// ====================
// c startup
// ====================
0: ld gp, 0(0)
1: st 0, 0(0)
2: lda fp, -0(gp)
3: lda sp, -0(gp)
4: push fp
5: lda 0, 2(pc)
6: push 0
7: ldc pc, 9
8: halt
// ====================
// main()
// ====================
9: lda sp, -2(sp)      // two local variables
10: ldc 0, 0           // number 0
11: st 0, -2(fp)       // l1 = 0
12: ldc 0, 1           // number 1
13: st 0, -3(fp)       // l2 = 1
14: ld 0, -2(fp)
15: out 0              // output l1
16: ld 0, -3(fp)
17: out 0              // output l2
18: ldc 27, 0          // return value = 0
19: lda sp, 0(fp)      // adjust sp
20: ld fp, 0(fp)       // restore old fp
21: ld pc, -1(sp)      // return
// ====================
```

3. Global-Variables 사용시

```
int g1;
int g2;

void main(void)
{
  g1 = 0;
  g2 = 1;
  output g1;
  output g2;
}
```

```
// ====================
// c startup
// ====================
0: ld gp, 0(0)
1: st 0, 0(0)
2: lda fp, -2(gp)    // 2 global variables
3: lda sp, -2(gp)
4: push fp
5: lda 0, 2(pc)
6: push 0
7: ldc pc, 9
8: halt
// ====================
// main()
// ====================
9: lda sp, -0(sp)   // zero local variables
10: ldc 0, 0        // number 0
11: st 0, -0(gp)    // g1 = 0
12: ldc 0, 1        // number 1
13: st 0, -1(gp)    // g2 = 1
14: ld 0, -0(gp)    // value of g1
15: out 0           // output g1
16: ld 0, -1(gp)    // value of g2
17: out 0           // output g2
18: ldc 27, 0       // preparing for return
19: lda sp, 0(fp)
20: ld fp, 0(fp)
21: ld pc, -1(sp)
// ====================
```

4. Array 형태로 된 Local Variable 사용시

```
void main(void)
{
  int l1;
  int l2[10];

  l1 = 0;
  l2[4] = 1;
  output l1;
  output l2[4];
}
```

```
// ====================
// c startup
// ====================
0: ld gp, 0(0)
1: st 0, 0(0)
2: lda fp, -0(gp)
3: lda sp, -0(gp)
4: push fp
5: lda 0, 2(pc)
6: push 0
7: ldc pc, 9
8: halt
// ====================
// main()
// ====================
9: lda sp, -11(sp)    // 11 local variables
10: ldc 0, 0          // number 0
11: st 0, -2(fp)      // l1 = 0
12: ldc 0, 4          // index
13: ldc 1, 1          // number 1
14: add 2, fp, 0      // fp + 4
15: st 1, -12(2)      //(fp + index) - 2 - n
16: ld 0, -2(fp)      // value of l1
17: out 0             // output l1
18: ldc 0, 4          // index
19: add 1, fp, 0      // fp + 4
20: ld 2, -12(1)      // (fp+index)-2-n
21: out 2             // output l2[4]
22: ldc 27, 0
23: lda sp, 0(fp)
24: ld fp, 0(fp)
25: ld pc, -1(sp)
// ====================
```

5. Expression-1 (register를 기반으로 한 연산)

```
int g1;

void main(void)
{
  g1 = 1 + 2 - 3;
  output g1;
  g1 = 1 - 2 * 3;
  output g1;
  g1 = (1 - 2) * 3;
  output g1;
  g1 = 1 + 2 / 3;
  output g1;
  g1 = (4 + 2) / 3;
  output g1;
}
```

6. Expression-2 (register를 기반으로 한 연산)

```
int g1;

void main(void)
{
  g1 = 1 > 2;
  output g1;
  g1 = 1 >= 2;
  output g1;
  g1 = 1 < 2;
  output g1;
  g1 = 2 <= 2;
  output g1;
  g1 = 1 == 2;
  output g1;
  g1 = 1 != 2;
  output g1;
}
```

7. If 문을 사용하는 경우

void main(void)
{
  int i;
  int j;
 	
  i = 10;
  j = 20;
 	
  if (i < j)
  output j;
  else
  output i;
}	

8. While 문 사용시

void main(void) {
  int i;
  int sum;
 	
  i = 1;
  sum = 0;
 	
  while (i <= 10) {
  sum = sum + i;
  i = i + 1;
  }
  output sum;
}	

9. Function Call (return 값이 없을 때)

int sum;
 	
int add(int x, int y)
{
  sum = x + y;
}
 	
void main(void)
{
  add(111, 222);
  output sum;
}
	// startup
	
	// add
	
	// main

10. Recursive Function Call과 Return 값이 있을 때

```
int sum(int n)
{
  int tmp;

  if (n == 1)
    return 1;
  else {
    tmp = sum(n - 1);
    return n + tmp;
  }
}

void main(void)
{
  int i;
  int j;

  i = 100;
  j = sum(i);
  output j;
}
```

// startup

// sum

// main