- 5 freedays -

Mobile processor Programming assignment #1: Simple calculator

모바일시스템공학과 32217259 문서영

1. 계산 순서에 따른 코드 설명

<main function>

1. input text 파일을 가져와 '₩n' 마다 읽는다. (교수님 코드 참고) 만약 생성된 파일이 잘못되었다면 error message 를 출력하고 종료되고, 잘 받았다면 변수 steam 을 사용하여 파일을 읽는다. 만약 파일이 없다면 종료되고, 파일이 있다면 while 문을 사용해 getline 함수로 읽은 line 이 종료될 때까지 한 줄씩 읽어나간다.

2. opcode, operand1, operand2 를 분리하기 위해 한 줄씩 읽은 코드를 strtok_r 함수를 사용한다. 여기서 만약 opCode 가 'H'라면 while 문을 끝내고 종료되도록 하였다.

<Get_Operation 함수>

3. operand1 과 oerand2 가 숫자(16 진수)인지, 레지스터인지 확인한 후 그에 맞는 숫자를 저장한다. op 의 값이 0x 로 시작한다면, 16 진수의 수이고, R 로 시작한다면 register number 값이다. 그래서 변수도 각각 다르게 regNum 과 num 을 사용하였고, strtol 함수를 사용하여 string 값을 숫자로 바꿔주었다.

```
// if op1 value is start with '0x'
num1 = strtol(op1, NULL, 16);
// if op2 value is start with 'R'
regNum2 = strtol(&(*(op2 + 1)), NULL, 10);
/* 원래 atoi 함수를 사용하려했지만, 입력했을 때 오류가 나와 ChatGPT 에 물어보니(+ 교수님 코드)
strtol()이 더 안전하다고 나와 바꾸게 되었다. */
```

(+, -, *, / 은 서로 유사한 연산이지만, 'M'은 operand1, operand2 에 register 만 들어올 수 있다. 또한 'J'와 'B'의 연산에서는 operand1 만 입력하고, 이 값이 10 진수라는 점을 확인한다.)

4. opcode 에 따라 연산 결과 저장을 다르게 한다. Switch-case 문을 사용하여 각 opCode 에 따라 나올 수 있는 값을 분리하였고, 각 case 안에서도 if 문을 사용해 Register 인지, number 인지에 따라 (register, register), (register, number), (number, number), (number, register) 4 가지의 경우로 또 다시 나눠 연산을 수행했다.

```
// '+' 연산에 대한 코드 (+, -, /, *의 코드 모두 연산기호만 달라짐)
case '+':
        if (*op1 == 'R') {
                if (*op2 == 'R') {
                        *(reg + regNum1) += *(reg + regNum2);
                         printf("R[%d]: %d = %s + %s\n", regNum1, reg[regNum1], op1, op2);
                }
                else {
                         *(reg + regNum1) += num2;
                          printf("R[%d]: %d = %s + %s", regNum1, reg[regNum1], op1, op2);
                }
        }
        else {
                if(*op2 == 'R'){
                         reg[0] = num1 + *(reg + regNum2);
```

```
                                        printf("R[%d]: %d = %s + %s\n", regNum1, reg[regNum1],
op1, op2);
                                }
                                else {
                                        reg[0] = num1 + num2;
                                        printf("R[%d]: %d = %s + %s\n", regNum1, reg[regNum1],
op1, op2);
                                }
                        }
                break;
```

'M' 연산은 또 다른 변수를 추가하여 register 값을 저장한 후, register 의 값을 옮긴 후에 op2 의 값을 0 으로 만들어주었다.

- 추가 연산
  - 'C'는 register 의 값을 비교하여 R0 에 -1, 0, +1 을 저장해주었다.
  - 'J' 연산은 J 뒤에 10 진수의 숫자의 줄로 가도록 새로운 JumpOp 함수를 만들어 사용하였다. JumpOp 에서는 lineNumber 값이 cnt (op1 의 값)보다 크거나 같을 때 읽도록 하였고, 나머지 동작은 main 함수와 같게 만들었다.
  - 'B' 연산은 위의 'J' 연산과 유사하게 작동하지만, R0 의 값이 0 일 때만 작동하도록 하였고, 0 이 아닐 때는 건너뛸 수 있도록 하였다.

5. 만약 opcode 에 'H'가 들어왔다면, 바로 연산을 종료한다. 위의 main 함수의 연산을 사용하여 프로그램을 종료한다. (R1 의 값을 출력 후, 종료 메시지와 함께 종료된다)

2. 어려웠던 점

- 파일 입출력을 통해 text 를 가져와 읽는 작업을 많이 안해봤기 때문에 교수님께서 코드를 보여주기 전까지는 코드 작성에 대한 어려움이 있었다.

<혼자 작성한 입출력 코드>

```
/* int Get_Operation function code */
        char getOP[20] = { 0 };
        char* opCode = NULL, * op1 = NULL, * op2 = NULL; // 부호, 실제 계산 숫자/reg 저장(2 개)
        int regNum1 = 0, regNum2 = 0; // store register number
        int num1 = 0, num2 = 0;
        long int result = 0;

        fgets(getOP, sizeof(getOP), stdin);
        char* temp = NULL;
        opCode = strtok_s(getOP, " ", &temp); //공백을 기준으로 문자열 자르기
        //opCode = strtok_r(getOP, " ", &temp);
```

<교수님 코드를 바탕으로 작성>

```
/* in main function code */
FILE* stream;
	char* line = NULL;
	size_t len = 0;
	ssize_t nread;
	char op;
	int lineNumber = 0;

	if (argc != 2) {
		fprintf(stderr, "Usage: %s <file>\n", argv[0]);
		exit(EXIT_FAILURE);
	}
	stream = fopen(argv[1], "r");
	if (stream == NULL) {
		perror("fopen");
		exit(EXIT_FAILURE);
	}


	while ((nread = getline(&line, &len, stream)) != -1){
		fwrite(line, nread, 1, stdout);

		// --- string setperation & start operation ---
		// ...

                                }
```

- assam 서버를 사용하여 vi 로 code 를 작성하고 있는데, wifi 오류로 인해 서버가 강제로 닫혔을 때 새로운 .nfs 라는 파일이 생겼다. 초반에는 위와 같이 .swp 파일이 생겨서 rm 으로 삭제하면 작동이 되었는데, 그 뒤로 네트워크 문제 등으로 인해 vi 파일이 강제 종료되어 .nfs 파일이 생성되었다. 이 파일은 'rm'을 사용하여 강제로 지워도 지워지지 않았다.

```
-rw-r--r-- 1 solid users 24576 Mar 26 09:05 .calcError.c.swp
-rw-r--r-- 1 solid users 12288 Mar 26 09:02 .nfs00000000088b2f3b0000000f
```

구글에 찾아봤을 때, lsof +D ./파일 (lsof - list open files)을 사용하여 현재 실행되고 있는 파일을 찾은 후, 그 파일의 Process ID 를 kill +9 PID 를 사용해 삭제할 수 있었다.

```
solid@solid-17259-5cc9fd6bff-2hf7j:~/hw1$ lsof +D ./
COMMAND    PID  USER   FD   TYPE DEVICE SIZE/OFF      NODE NAME
bash    275705 solid  cwd    DIR  0,172      218 133301869 .
vi      275711 solid  cwd    DIR  0,172      218 133301869 .
vi      275711 solid    4u   REG  0,172    12288 143339323 ./.nfs00000000088b2f3b0000000f
bash    275762 solid  cwd    DIR  0,172      218 133301869 .
lsof    275778 solid  cwd    DIR  0,172      218 133301869 .
lsof    275779 solid  cwd    DIR  0,172      218 133301869 .
```

위의 사진을 확인하면 .nfs 파일이 잘 삭제된 것을 볼 수 있다.

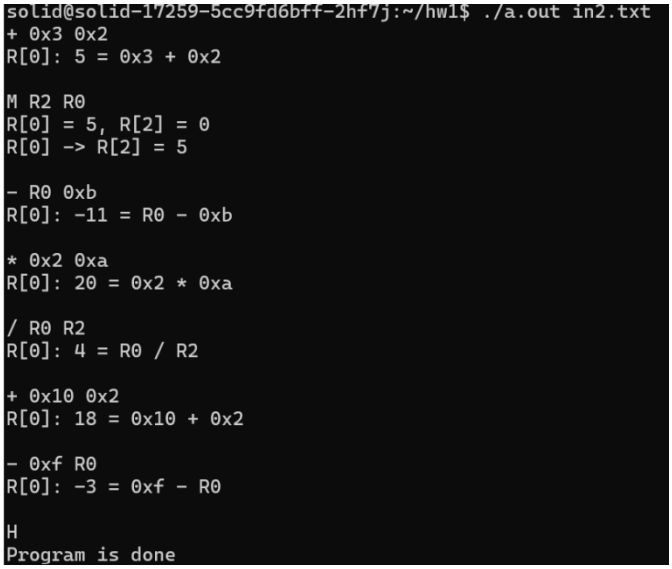
- jump 연산을 만들 때, 또 다른 함수를 만들어 사용해야 할지, 같은 함수에서 계속 이어가는 게 맞을지 고민하다가 새로운 함수 JumpOp 를 만들었다. 처음에는 문제를 잘못 이해하여 현재 text 파일이 아닌 다른 text 파일을 열어서 연산을 이어가는 거라고 착각하였지만, 착각한 부분을 인지하고 만들게 되었다.

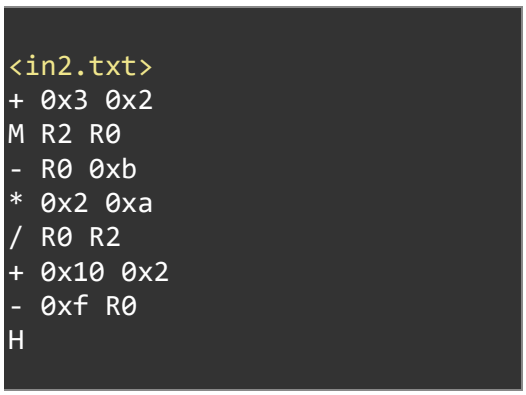
3. ChatGPT 사용한 부분

1. 파일 입출력 실행 방법과 string 을 숫자로 바꾸는 방법을 찾아보았다. (strtol 함수)
2. GCD 연산을 만들 때, 연산 결과를 정확하게 구하기 위해 사용했다.
3. jump operation 에서 파일의 처음부터 다시 읽어야 하는데, 관련 함수를 찾기 위해 사용하였다. (fseek 함수)

4. 실행 결과 예시

1> just four fundamental arithmetic operations





2> GCD 96, 15 (result store at R1)

```
+ R1 0x60
+ R0 0x0F
B 15
M R4 R4
* R2 0x0
* R3 0x0
+ R2 R1
+ R3 R1
M R1 R0
/ R2 R1
* R2 R1
- R3 R2
M R0 R3
J 3
H
```

<Result>

```
solid@solid-17259-5cc9fd6bff-2h
+ R1 0x60
R[1]: 96 = R1 + 0x60
+ R0 0x0F
R[0]: 15 = R0 + 0x0F
B 15
r[0] = 15
M R4 R4
R[4] = 0, R[4] = 0
R[4] -> R[4] = 0

* R2 0x0
R[2]: 0 = R2 * 0x0

* R3 0x0
R[3]: 0 = R3 * 0x0

+ R2 R1
R[2]: 96 = R2 + R1

+ R3 R1
R[3]: 96 = R3 + R1

M R1 R0
R[0] = 15, R[1] = 96
R[0] -> R[1] = 15

/ R2 R1
R[2]: 6 = R2 / R1

* R2 R1
R[2]: 90 = R2 * R1

- R3 R2
R[3]: 6 = R3 - R2

M R0 R3
R[3] = 6, R[0] = 0
R[3] -> R[0] = 6

J 3
go to 3 line
B 15
r[0] = 6
* R3 0x0
R[3]: 0 = R3 * 0x0

M R1 R0
R[0] = 6, R[1] = 15
R[0] -> R[1] = 6

- R3 R2
R[3]: -90 = R3 - R2

H
reg[1] is 6. Program is done
```

GCD 연산은 재귀함수를 응용하여 만들었고, Result 값은 R1 에 저장하도록 하였다. 또한 GCD 96, 15 가 아니라 16 진수로 변환된 값 0x60, 0xF 의 값을 사용하였다.

5. 사용 도구(툴, 언어)

C 언어를 사용했으며 툴은 초반에 코드 틀을 사용할 땐 Visual Studio 2022 을 사용하였고, 파일 I/O 를 확인하고 전체적인 내용을 수정할 때는 assam server 와 온라인 C Compiler 를 사용하였습니다.