Mobile processor Programming

assignment #1: Simple calculator

모바일시스템공학과 김태경 (32211203)

목차

2024.03.31

1. 프로젝트 목표

이 과제는 기본적인 연산들의 계산과, 컴퓨터의 언어로 계산을 수행하는 방법과 레지스터 명령어의 기능을 수행시켜 봄으로서 컴퓨터가 일반적인 컴퓨터의 명령을 실행하는 과정을 이해하는 것이다.

2. 프로젝트 조건

기본요구조건
- 기본적인 연산인 '덧셈', '뺄셈', '곱셈', '나눗셈'의 작동을 구현한다.
- 피연산자는 16진수 또는 레지스터 번호일 수 있으며, 16진수는 '0x', 레지스터는 'R'을 접두사로 가진다.
- 결과값의 저장위치는 레지스터 번호 0번인 'R0'에 저장된다.
- 16진수 또는 레지스터에서 레지스터로 값을 이동시키는 'M 연산'을 지원한다.
- 예외처리 '.txt파일 불러오기', '나눌 값이 0' 등 예외가 발생할 수 있는 지점을 보완한다.
- 'inst_reg'라는 새 변수를 정의하여 파일에서 명령어를 가져온다.

추가기능
- 비교연산(Compare): 'C' 명령어를 사용하여 두 레지스터의 값을 비교, 결과를 'R0'에 저장
- 점프연산(jump); 'J' 명령어를 사용하여 특정 입력 명령어로 점프할 수 있다.
- 조건부 점프연산(Branch_Equal): 'BEQ' 명령어를 이용하여 'R0'의 값이 0일 경우 목표 명령어로 점프할 수 있다.
- 최대공약수(Greatest common divisor): 'GCD' 명령어를 이용하여, 두 숫자의 최대공약수를 계산할 수 있다.

3. 프로그램 빌드 환경

코딩 검증 환경: Visual Studio(2024)
컴파일 환경: Linux Assam
컴파일 방법: gcc simple.c
실행 방법: ./a.out

4. 프로그램 구성

0) 전역변수 선언

```
int regs[10] = { 0 };
char inst_reg[MAX_LINE_LENGTH];
```

- inst_reg: 입력파일의 내용 중 한 줄을 저장할 배열 선언
- regs[10]: 임의로 지정한 각 값들이 들어갈 레지스터의 배열 선언

1-1) main함수 1

```
int main() {
    FILE* fp;

    fp = fopen("input.txt", "r");
    if (fp == NULL) {
        perror("오류 : input.txt파일이 존재하지 않음. \n");
        exit(EXIT_FAILURE);
    }
```

- ‘input.txt‘ 입력파일 열기 및 예외처리

1-2) main함수 2

```
    while (fgets(inst_reg, MAX_LINE_LENGTH, fp) != NULL) {
        char* op = strtok(inst_reg, " ");
        char* op1_str = strtok(NULL, " ");
        char* op2_str = strtok(NULL, " \n");

        int op1, op2;
        if (op1_str[0] == 'R') {
            op1 = regs[op1_str[1] - '0'];
        }
        else {
            op1 = strtol(op1_str, NULL, 16);
        }

        if (op2_str[0] == 'R') {
            op2 = regs[op2_str[1] - '0'];
        }
        else {
            op2 = strtol(op2_str, NULL, 16);
        }
```

- ’fgets‘ 함수를 사용하여 입력파일에서 한 줄씩 문자열을 읽어온다.
- ’strtok‘ 함수를 사용하여 연산기호와 입력값1, 입력값2를 분리한다.
- 입력값이 16진수의 형태인지 레지스터인지 판별하고 정수로 변환한다.

1-3) main함수 3

```
        if (strcmp(op, "+") == 0) {
            add(op1, op2);
        }
        else if (strcmp(op, "-") == 0) {
            sub(op1, op2);
        }
```

- 조건문 if를 이용한 명령어 확인 및 명령어에 맞는 함수를 이용한 계산

```
    fclose(fp)

    return 0;
}
```

- 열어둔 입력파일 닫기 & 함수 종료

2-1) 기본연산 '+', '-', '*', '/'

```
// 덧셈
void add(int op1, int op2) {
    regs[0] = op1 + op2;
    printf("R0: %d = %d + %d\n", regs[0], op1, op2);
}
```

- 각각 연산자 기호에 따른 결과값은 0번 레지스터에 저장 및 계산과정 출력

뺄셈, 곱셈 동일

```
// 나눗셈
void divide(int op1, int op2) {
    if (op2 == 0) {
        printf("오류: 0으로 나눌 수 없음\n");
        return;
    }
    regs[0] = op1 / op2;
    printf("R0: %d = %d / %d\n", regs[0], op1, op2);
}
```

- 나눗셈의 경우 0으로는 나눌 수 없으므로 예외처리

2-2) 추가 기능 'M', 'C', 'J', 'BEQ', 'GCD'

```
// M 연산
void M(int op1, int op2) {
    regs[op2] = op1;
    printf("R%d: %d", op1, op2);
}
```

- 'M' 함수는 op1 값을 op2 레지스터에 저장한다.

```
// C 연산 (Compare)
void C(char* op1_str, char* op2_str) {
    int res;
    int op1 = strtol(op1_str, NULL, 16);
    int op2 = strtol(op2_str, NULL, 16);

    if (op1 == op2)
        res = 0;
    else if (op1 > op2)
        res = 1;
    else
        res = -1;

    regs[0] = res;
    printf("R0: %d\n", res);
}
```

- 'C' 함수는 입력된 문자열을 정수 값으로 변환하고, 두 값을 비교하여 결과를 레지스터에 저장한다.

```
// J 연산 (Jump)
void J(int target_I) {
    printf("Jump to instruction %d\n", target_I);

}
```

- 'J' 함수는 주어진 목표 명령어로 점프하는 역할을 한다.

```
// BEQ 연산 (Branch_Equal)
void B(int target_I) {
    if (regs[0] == 0) {
        printf("Branch to instruction %d\n", target_I);

    }
}
```

- 'B' 함수는 이전 비교 결과가 0인 경우에만 주어진 목표 명령어로 점프하는 역할을 한다.

```c
// GCD 연산 (Greatest common divisor)
void G(int a, int b) {
    int i, gcd;
    int s = (a < b) ? a : b;
    for (i = 1; i <= s; i++) {
        if ((a % i == 0) && (b % i == 0)) {
            gcd = i;
        }
    }
    regs[0] = gcd;
    printf("R0: %d \n", regs[0]);
}
```

- 'G' 함수는 두 정수를 유클리드 알고리즘을 사용하여 최대공약수를 계산하고, 계산된 최대공약수를 레지스터 0에 저장한다.

3) 입력파일 생성

```
solid@taegyung21-b694b9fc5-8kjs5:~/test_prog$ vi input.txt
solid@taegyung21-b694b9fc5-8kjs5:~/test_prog$ cat input.txt
+ 0x3 0x2
- 0x1 0x2
* R1 R2
/ 0x8 0x2
M 0x5 R3
M R4 R5
C R6 R6
GCD 15 96
J 3
BEQ 7
```

- input.txt 파일 생성 및 계산할 문제 입력

4) 컴파일 및 실행결과

```
+   solid@taegyung21-b694b9fc5-8kjs5:~/test_prog$ ./a.out
-   R0: 5 = 3 + 2
*   R0: -1 = 1 - 2
/   R0: 0 = 0 * 0
M   R0: 4 = 8 / 2
M   R2: 4
C   R7: 0
GCD R0: 0
J   R0: 3
BEQ Jump to instruction 3
    Jump to instruction 7
    Segmentation fault (core dumped)
```

5. 문제점 및 해결방법

1) 리눅스 환경에서의 C언어 프로그래밍에 있어 오류의 확인과 조작미숙으로 익숙하지 않은 환경에서의 코딩에 시간소모를 느낌
-> 기존에 C언어를 연습하던 Visual Studio에서 코드를 완성하고 리눅스 서버에 옮겨와 실행하는 방식으로 과제 수행시간을 단축함

2) 비교연산을 하기 위한 'C'함수를 작성하던 중 결과값이 나오지 않는 상황이 발생

```c
// C 연산 (Compare)
void C(int op1, int op2) {
    if (op1 == op2) {
        regs[0] = 0;
    }
    else if (op1 > op2) {
        regs[0] = 1;
    }
    else {
        regs[0] = -1;
    }
}
```

(수정 이전)

```c
// C 연산 (Compare)
void C(char* op1_str, char* op2_str) {
    int res;
    int op1 = strtol(op1_str, NULL, 16);
    int op2 = strtol(op2_str, NULL, 16);

    if (op1 == op2)
        res = 0;
    else if (op1 > op2)
        res = 1;
    else
        res = -1;

    regs[0] = res;
    printf("R0: %d\n", res);
}
```

(수정 이후)

-> 이전의 경우는 입력값1, 입력값2를 레지스터의 형태를 신경쓰지 않고 바로 정수(int)로 입력 받는다고 생각하여, 함수의 매개변수를 정수로 받아서 결과값이 나오지 않았으나, 문자형(char)로 받아 정수형으로의 변환 후 계산함으로서 올바른 결과값이 출력되도록 수정함

3) 나눗셈의 함수원형 정의와 함수정의에 있어서 divide의 축약으로 'div'를 함수명으로 정의했는데 Visual Studio에서 이에 대한 오류를 반환함
-> 'div'는 이미 <stdlib.h> 라이브러리에 몫과 나머지의 연산을 수행하는 함수로 이미 정의되어 있어 함수명으로서 사용할 수 없다. 따라서 함수명을 'divide'로 사용

6. 느낀점

프로그램을 작성하면서 느낀 것은 리눅스 환경에 아직 익숙하지 않아, 프로그래밍 중 조작 미숙으로 불편함을 많이 느꼈으며, 기한 내에 맞추기 위해 도중에 Visual Studio로 넘어와 코드를 완성하고 리눅스 서버로 옮기기로 결정한 것이 다행이라고 생각한다. 다만 리눅스의 man_page 기능은 기존의 프로그래밍을 하는 방법은 잘 모르는 용어에 대해서 정확한 정의를 알 수 있어서 man_page의 내용을 기반으로 인터넷으로 사용방법과 응용방법, 예제 찾기에 시간을 아낄 수 있었다.

프로그래밍의 구성에 대하여 컴퓨터가 이해하기 위해 만들어진 어셈블리 언어를 이용한 계산을 구현하기 위해 16진수와 레지스터 번호를 임의로 레지스터 배열을 만들고, 그 배열 내부에서 컴퓨터의 PC, 레지스터, 메모리의 이용하는 방식을 유사하게 구현하려 한 것 같은데 MIPS의 일부 명령어만 이해하고 있다는 생각이 들어 다른 자료를 더 찾아볼 생각이다.