

컴퓨터구조(4471029)

실습2 - MIPS Instruction의 이해

서론

이번 실습에서는, MIPS 명령어를 실행할 수 있는 에뮬레이터인 QtSpim을 활용하여, MIPS 어셈블리어 프로그램을 직접 디버깅하는 과정을 진행합니다. 이를 통해 MIPS CPU의 구조와 동작에 대해 이해할 수 있습니다. 실습2가 완료된 후, 실습 Quiz(Q1, Q2, ...)의 답안을 이루리 시스템 내 실습#2에 입력 후 제출합니다. 실습2의 답안을 전부 모아 텍스트로 제출하면 되겠습니다. **기타 사항은 공지사항 내 “실습 결과 제출” 게시물을 참고 바랍니다. 실습 결과 제출 기간이 지정되어 있으므로, 반드시 이루리 시스템에 접속 후 제출 마감기한을 확인바랍니다.**

참고자료

QtSpim: <https://sourceforge.net/projects/spimsimulator/files/>

SPIM이란, MIPS 프로세서를 시뮬레이션할 수 있는 소프트웨어입니다. 실제 MIPS 프로세서보다는 훨씬 느리지만, 디버깅이나 교육의 목적으로 많이 사용됩니다. 이 중, 현재는 QtSpim이 많이 사용되고 있습니다. QtSpim은 Qt GUI 프레임워크를 사용한, 오픈소스 SPIM 에뮬레이터입니다. QtSpim을 이용해 MIPS32 어셈블리 언어 소스코드 파일 (.asm)을 읽어 실행해볼 수 있습니다. 실행 도중 레지스터나 메모리의 데이터를 직접 읽거나 수정하는 등의 디버깅 작업이 가능합니다. QtSpim은 Windows/Linux/macOS(OS X)에서 동작합니다([참조: 공식 홈페이지](#)).

목표

- QtSpim을 이용하여, 주어진 MIPS 어셈블리어 소스 코드를 읽어 실행합니다.
- 코드를 분석해 보고, 단계별로 실행하며 실습을 진행합니다.

실습

1. QtSpim 프로그램을 실행합니다.

2. 실습에 맞는 시뮬레이터 설정을 진행합니다.


(이전 과제에서 진행된 부분이기 때문에 넘어가도 무관한 부분입니다.)

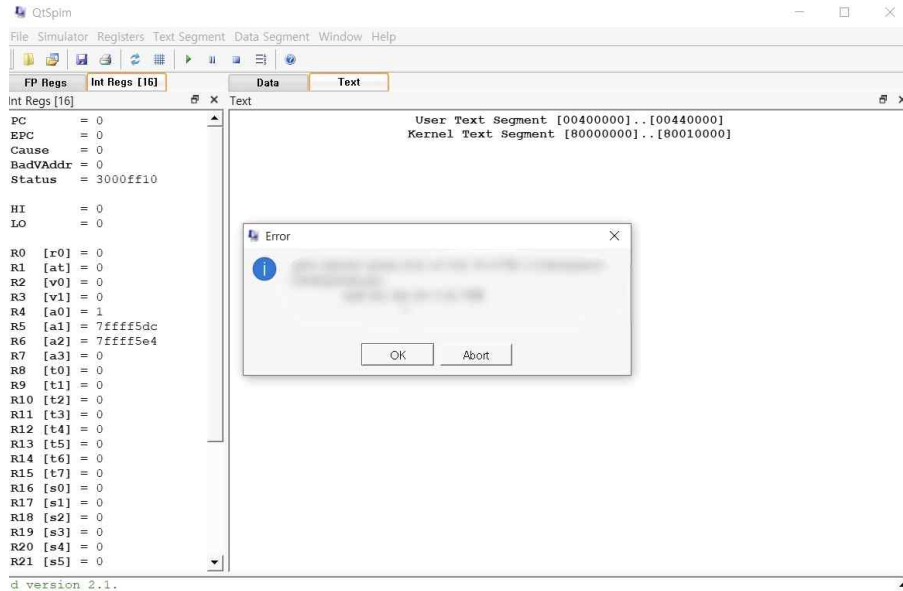
- (1) 상단 “Simulator → Settings”에서 아래 설정을 진행합니다.
- (2) “MIPS” 탭에서, “Simple Machine”을 클릭해, 상단 5개의 체크박스 중 “Accept Pseudo Instruction”만 체크되도록 합니다.
- (3) 아래 “Load Exception Handler”를 체크 해제합니다. 과정이 완료되면, OK를 눌러 저장합니다.
- (4) 상단 “Simulator → Run Parameters”에서 아래 설정을 진행합니다.
- (5) “Address or label to start running program”을 **0x00400000** 으로 설정하고, OK를 눌러 저장합니다.

※ 프로그램의 시작점은 첫 명령어를 로드할 위치를 나타냅니다.

앞으로의 실습에서의 코드 시작점은 항상 **0x00400000** 으로 생각하고 진행합니다.

실습2-1: Debugging MIPS assembly code

3. File 메뉴에서,  Reinitialize and Load File 을 클릭하여 task.asm 을 로드합니다. 이 때, 파일이 로드되지 않고 오류가 발생합니다. 오류 창에는 task.asm의 어떤 부분에서 오류가 발생했는지 나타납니다.





task.asm은 사용자로부터 두 개의 값을 키보드로 입력받아, 이 두 값의 합을 출력하는 프로그램입니다. 이를 확인한 뒤, task.asm을 텍스트 에디터로 열어, 코드를 수정합니다.

Q1. 코드의 어느 부분을 수정해야 프로그램이 정상적으로 작동하는지 작성하세요.

답 : _____ → _____

Q2. 오류가 발생한 이유는 무엇인지 명령어에 대한 설명과 함께 서술하세요.

답 : _____

4. task.asm을 수정하고, 다시 File 메뉴에서  Reinitialize and Load File 을 클릭하여 task.asm을 로드합니다.
정상적으로 로드가 되면 Text Segment에 12줄 정도가 추가됩니다. (Console에는 아무것도 없는 상태.)
5. 로드된 프로그램을 실행 버튼  을 눌러 실행합니다. 그 다음, Console창에 두 개의 값을 입력하고 두 값의 합이 아래 예시와 동일하게 출력되는지 확인합니다.



SPIM은 시스템 호출 명령을 통해 아래와 같은 운영체제 서비스를 제공합니다. 예를 들어, `li $v0, 5` 명령어의 경우 5(system call code)를 호출해 와서 int 형식의 값을 키보드로 읽어와 저장하게 됩니다. 실습을 진행하는 데 있어 필요시 다음의 ¹⁾system call code table을 참고하세요.

| Service | System call code | Arguments | Result |
|--------------|------------------|---|-----------------------------|
| print_int | 1 | \$a0 = integer | |
| print_float | 2 | \$f12 = float | |
| print_double | 3 | \$f12 = double | |
| print_string | 4 | \$a0 = string | |
| read_int | 5 | | integer (in \$v0) |
| read_float | 6 | | float (in \$f0) |
| read_double | 7 | | double (in \$f0) |
| read_string | 8 | \$a0 = buffer, \$a1 = length | |
| sbrk | 9 | \$a0 = amount | address (in \$v0) |
| exit | 10 | | |
| print_char | 11 | \$a0 = char | |
| read_char | 12 | | char (in \$v0) |
| open | 13 | \$a0 = filename (string), \$a1 = flags, \$a2 = mode | file descriptor (in \$a0) |
| read | 14 | \$a0 = file descriptor, \$a1 = buffer, \$a2 = length | num chars read (in \$a0) |
| write | 15 | \$a0 = file descriptor, \$a1 = buffer, \$a2 = length | num chars written (in \$a0) |
| close | 16 | \$a0 = file descriptor | |
| exit2 | 17 | \$a0 = result | |

1) Computer Organization and Design, Appendix A-44, Figure A.9.1 System Services.

실습2-2: Array manipulation

6. task.asm 실습을 마쳤으면, task2.asm 파일에 마우스 오른쪽 클릭을 하여 연결 프로그램 - 메모장을 이용하여 파일을 열어줍니다. 파일을 열어보면 다음과 같은 코드를 확인할 수 있습니다.

```
task2.asm
1  # task2.asm
2  .data
3  myArray: .word 10, 5, 10, 5
4
5  .text
6  main:   la $s1, myArray
7          li $s2, 12
8
9          # place your code here!
10         # place your code here!
11         # place your code here!
12
13         li $v0, 1
14         move $a0, $t0
15         syscall
16
17  exit:   li $v0, 10
18         syscall
19
```



실습에 들어가기 앞서, 줄 별로 코드를 살펴봅시다.

- | | |
|------------|---|
| [3] | myArray 배열을 선언하고 값을 넣어줍니다. |
| [6] | s1 레지스터를 myArray 배열의 시작주소로 지정합니다. |
| [7] | s2 레지스터에 숫자 12(h값)를 저장합니다. |
| [9,10,11] | 아래의 문제를 통해 해당 코드를 작성하세요. |
| [13,14,15] | \$t0에 저장된 값을 출력합니다. (3페이지의 system call code 1 참고) |
| [17,18] | 프로그램을 종료합니다. |

- Q3. 밑의 코드는 [9,10,11] 줄에 해당하는 C언어 코드입니다. 다음의 코드를 어셈블리어 코드 3줄로 변환하여 작성하세요. (*배열 A의 시작 주소는 \$s1에 기억되며, 변수 h는 \$s2에 대응됩니다.)

C언어 코드: $A[3] = h + A[2]$

답: _____


7. Q3을 통해 코드를 작성했다면 파일을 저장한 후 File -  Reinitialize and Load File 을 통해 저장한 task2.asm 파일을 불러온 후  을 눌러 실행시켜 줍니다. 그리고 console창을 확인해 봅니다.


Q4. 파일을 실행시키면 console 창에 특정 값이 뜨는 것을 확인할 수 있습니다. 왜 해당 값이 console 창에 뜨는지 Q3에 작성한 코드의 동작 원리와 관련지어 설명하세요.

console 창에 뜬 값: _____

설명: _____


실습2-3: Loop over array

8. File 메뉴에서,  Reinitialize and Load File 을 클릭하여 task3.asm을 로드합니다.

로드된 프로그램을  Run/Continue 버튼을 클릭해, 실행해봅니다. 결과는 아래와 같이 나타납니다:



이를 통해, task3.asm 파일은 배열 myArray에 저장된 값들을 순서대로 출력해주는 프로그램임을 알 수 있습니다. myArray의 값을 바꿔 실행해보면, 그대로 화면에 출력되는 것을 확인할 수 있습니다.

File 메뉴에서,  Reinitialize and Load File 을 클릭하여, 프로그램을 다시 로드합니다. 이 때, 프로그램을 한 줄씩 실행해봅니다. 필요시 반복해서 다시 로드하여 실행하여도 됩니다.

- Q5. 이 프로그램에서는 System call 관련 변수를 제외하면, 변수 \$t0~\$t3까지가 주로 사용됩니다.

각 변수들에 저장된 값들이 하는 역할을 간단히 설명하세요.

1. \$t0 (Line 10): _____
2. \$t1 (Line 11): _____
3. \$t2 (Line 14): _____
4. \$t3 (Line 17): _____

※ 문제 해결 시, QtSpim 외에 메모장이나 VSCode와 같은 에디터를 같이 띄워 병행하면, 코드를 분석하는 데에 도움이 될 수 있습니다. 두 개를 함께 보면서 진행하는 것을 추천합니다.

9. task4.asm 파일을 텍스트 에디터로 열어 내용을 확인합니다.

새로운 프로그램 task4의 목적은, 사용자로부터 정수 8개를 읽어 배열 myArray에 저장한 뒤, 전부 읽은 후 이를 차례대로 다시 출력합니다. 이 프로그램을 구현하여 실행한 결과는 아래와 같습니다.

```
7
24
1312
43
675
-99
2328
0
7 24 1312 43 675 -99 2328 0
```

task4.asm은, 기존의 task3.asm 파일에서의 코드를 재사용하여 구현했습니다. 추가적으로 사용자로부터 값을 입력받는 system call은 아래 system call code table을, 메모리 저장/로드 명령어는 아래 opcode table을 활용하여 작성할 수 있습니다.

| Service | System call code | Arguments | Result |
|--------------|------------------|---|-----------------------------|
| print_int | 1 | \$a0 = integer | |
| print_float | 2 | \$f12 = float | |
| print_double | 3 | \$f12 = double | |
| print_string | 4 | \$a0 = string | |
| read_int | 5 | | integer (in \$v0) |
| read_float | 6 | | float (in \$f0) |
| read_double | 7 | | double (in \$f0) |
| read_string | 8 | \$a0 = buffer, \$a1 = length | |
| sbrk | 9 | \$a0 = amount | address (in \$v0) |
| exit | 10 | | |
| print_char | 11 | \$a0 = char | |
| read_char | 12 | | char (in \$v0) |
| open | 13 | \$a0 = filename (string), \$a1 = flags, \$a2 = mode | file descriptor (in \$a0) |
| read | 14 | \$a0 = file descriptor, \$a1 = buffer, \$a2 = length | num chars read (in \$a0) |
| write | 15 | \$a0 = file descriptor, \$a1 = buffer, \$a2 = length | num chars written (in \$a0) |
| close | 16 | \$a0 = file descriptor | |
| exit2 | 17 | \$a0 = result | |

MIPS system call code table

| Instruction | Example | Meaning | Comments |
|-------------|-------------------|--------------------------|------------------------------|
| load word | lw \$s1, 20(\$s2) | \$s1 = Memory[\$s2 + 20] | Word from memory to register |
| store word | sw \$s1, 20(\$s2) | Memory[\$s2 + 20] = \$s1 | Word from register to memory |

MIPS load/store word code table

Q6. task4.asm에서, 비어있는 20~22번째 줄에 MIPS instruction을 구현함으로써 코드를 완성할 수 있습니다. 아래 비어있는 칸에 입력할 코드를 서술하세요.

```
19 # save into array body
20  # system call for read_int
21  # read integer
22  # copy arguments to array
23
24 bne $t1, $t2, loadloop # branch if loop is end (array boundary)
```