# Computer Organization Assignment#1 MIPS ISS(Instruction Set Simulator) Due : 23:59, Monday, October 8, 2012

### I. Introduction

ISS를 사용하여 MIPS instruction을 익숙하게 함이 목적이다. ISS를 사용하면 각 instruction이 어 떤 동작을 하며 어떻게 메모리에 접근하여 프로그램을 수행하는지를 눈으로 확인할 수 있을 뿐만 아니라, 32개의 GPR(General Purpose Register)의 역할을 확인하고 직접 컨트롤이 가능하다. 크게 3가지 과제를 수행하게 되는데 첫째로, MIPS ISS중 하나인 SPIM의 사용법을 익히고, 둘째로, 35개 의 instruction에 대해 그 동작을 익힌다. 셋째로, 공부한 instruction을 바탕으로 직접 assembly coding을하고 ISS를 통하여 simulation 해 본다. ISS가 무엇인지, assembly language는 무엇인지에 대한 기본적인 개념은 본 문서 맨 뒤의 APPENDIX A에 첨부하였다.

### II. Theory

1. SPIM의 사용 법

	🧏 QtSpim				at last 10	_	-			x
	File Simula	ator <u>R</u> egisters	Text :	Segment <u>D</u> ata	Segment <u>W</u> indov	v <u>H</u> elp				
	🖻 🖉		ı 🏢	• • • •	. Ξ) Ø					
	Regs I	nt Regs [16]	∢►	Data	Text					
PC &	nt Regs [16]		₽×	Data 🔪	· · ·					₽×
MIPS registres	PC	= 0								*
init bregistres	EPC	= 0		Kernel dat	a segment [90	000000]. [	90010000]			
	Cause	= 0		[90000000]	78452020	74706563	206e6f69	636f2000	Exception . oc	
	BadVAddr	= 0		[90000010]	Data	49562020	Progra	72616667	curred and ignor	
	Status	= 3000ff10		[9000003N	- 66020405	4c545b20	20005d42	4c545b20	1. (TLB). (TL	
	нт	= 0		[90000040]	20005d42	40545020	20005d42	64415b20	B]. [TLB]. [Ad	
	LO	= 0		[90000050]	73657264	12652073	20726172	69206e69	dress error in i	
				[90000060]	2f74736e	61746164	74656620	205d6863	nst/data fetch]	
	R0 [r0]	= 0		[90000070]	5b202000	72646441	20737365	6f727265	. [Address erro	
	R1 [at]	= 0		[90000080]	6e692072	6f747320	205d6572	56202000	r in store]. [	
	R2 [v0]	= 0		[90000090]	20646142	/4/36669	/463/5/2	20686169	Bad instruction	
	R3 [v1]	= 0		[900000000]	74616420	64612061	20200020	00205473	data addreeel	
	R4 [a0]	= 3		[900000c0]	455b2020	726f7272	206e6920	63737973	[Error in svsc	
	R5 [a1]	= 7ffff008	=	[000000d0]	5d6c6c61	20200020	6572425b	6f706b61	all]. [Breakpo	
	R7 [a3]	= 0	-	[900000e0]	5d746e69	20200020	7365525b	65767265	int]. [Reserve	
	R8 [t0]	= 0		[900000f0]	6e692064	75727473	6f697463	00205d6e	d instruction] .	
	R9 [t1]	= 0		[90000100]	5b202000	74697241	74656d68	6f206369	. [Arithmetic o	
	R10 [t2]	= 0		[90000110]	66726576	5d776f6c	20200020	6172545b	verflow]. [Tra	
	R11 [t3]	= 0		[90000120]	00205d70	56202000	616f6c46	676e6974	p] [Floating	
	R12 [t4]	= 0		[90000130]	696I7020	2050/466	20000000	61435020	pointj [Co	
	R13 [t5]	= 0		[90000140]	00545844	575b2020	20000000	2020005d	MY1 [Watch]	
	R14 [t6]	= 0		[90000160]	63614d5b	656e6968	65686320	005d6b63	[Machine check].	
	R15 [t7]	= 0		[90000170]	00000000	5b202000	68636143	00005d65	[Cache]	
	R17 [s1]	= 0		[90000180]	90000024	90000033	900003b	90000043	\$ 3 ; C	
	R18 [s2]	= 0		[90000190]	9000004b	90000071	9000008d	900000aa	Кq	
	R19 [s3]	= 0		[900001a0]	90000c0	900000d6	900000e6	90000100		
	R20 [s4]	= 0		[900001b0]	90000101	9000011a	90000124	90000125	\$ *	
	R21 [s5]	= 0		[900001c0]	90000139	9000013a	9000013b	90000148	9	
	R22 [s6]	= 0		[90000140]	90000149	9000014a	90000146	90000154	1JKT	E
	R23 [s7]	= 0		[90000160]	90000132	90000170	90000171	90000172	• + »	
	R24 [t8]	= 0		[90000200]	[9000ffff]	00000000	500001/0	50000171	5	
	R25 [19]	= 0								
	R27 [k1]	= 0	-							-
	SPIM Versio	n 414 of Sent	emher	· 4 - 2011						
	Copyright 19	990-2010, Jame:	s R, L	arus,						^
	All Rights R	eserved,								
	SPIM is dist	tributed under a	a BSD full ⇔	license,						
	Jee uie lile	THEADINE TOP a	iun C	opyngni nodce,						-
			_				_			

Figure 1 SPIM window layout

여타 ISS와 달리 SPIM은 단순히 instruction을 수행해 볼 수 있을 뿐만 아니라 console을 통해 가 상으로 수행결과를 출력한다든지(printf)의 'systemcall'명령이 존재한다. 단순히 processor만 있고 OS가 없다면 프로세서가 수행 한 instruction의 결과를 눈으로 확인하기 어려운데, SPIM에서는 몇 가지 자체 systemcall 을 사용하여 여러 가지 기능을 구현할 수 있다. 뿐만 아니라 가상적으로 Memory map을 구성하고 있기 때문에 stack, data sement 등의 분리 된 memory 영역에 접근하 고 프로그램 수행에 이용할 수 있다. 이러한 systemcall은 SPIM에서만 유효한 것이기 때문에 이를 이용하기 위한 몇 가지 약속을 알아야 한다.

SPIM example 1: add two numbers							
# \$t2 - used to hold the sum of t	# \$t2 - used to hold the sum of the \$t0 and \$t1.						
# \$v0 - 이 레지스터는 어떤 syste	m call을 수행할 것인지 명령을 받는 역할로 정해져 있다.						
# \$a0 - 이 레지스터는 특정 syste	m call에서 input을 받는 경우 input parameter로 정해져 있다.						
. <b>text</b> # 여기서부터 code가	시작한다는 directive						
main:	#'main' 이라는 label						
li \$v0, 4	# read number into \$v0						
la \$a0, str1	# str1 label 주소로 \$a0에 저장						
syscall	# str1 print						
li \$v0, 5	# read number into \$v0						
syscall	# make the syscall read_int						
move \$t0, \$v0	# move the number read into \$t0						
li \$∨0, 4							
la \$a0, str2							
syscall	# str2 print						
li \$v0, 5	# read second number into \$v0						
syscall	# make the syscall read_int						
move \$t1, \$v0	# move the number read into \$t1						
add \$t2, \$t0, \$t1							
li \$v0, 4							
la \$a0, str3							
syscall	# str3 print						
move \$a0, \$t2	# move the number to print into \$a0						
li \$v0, 1	# load syscall print_int into \$v0						
syscall	# print \$a0 to console						
li \$v0, 10	# syscall code 10 is for exit						
syscall							
.data							
str1 : .asciiz "Input 1 :"							
str2 : .asciiz "Input 2 :"							

str3 : .asciiz "Result :"

```
- #는 주석을 나타낸다.
```

- main : 은 label을 나타낸다. jump 등을 할 때 label을 명시하면 label로 jump 한다.

앞의 예제는 \$t0 와 \$t1에 들어갈 값을 사용자로부터 console창을 통해 입력 받아서 그 합을 출 력하는 assemply code이다. 여기서 .text, syscall, move, li 등 실제 MIPS instruction에는 포함되지 않은 psudo code와 유사한 개념으로 assemply 코딩을 쉽게 하도록 돕고 있다. 이에 대한 설명을 아래에 보충한다.

아래의 Table1은 SPIM에서 사용되는 directives로서 assemply 코딩을 할 때 필요하다. 사용 예는 example1을 참조한다.

### Table 1 SPIM directives

.data	start data segment
.ascii str	store the string str in memory without $\langle 0 \rangle$
.asciiz str	idem, with '\0'
.byte 3,4,16	store 3 byte values
.double 3.14, 2.72	store 2 doubles
.float 3.14, 2.72	store 2 floats
.word 3,4,16	store 3 32-bit quantities
.space 100	reserve 100 bytes
.text	start text segment

#### Table 2 syscall services

Service	명령	INPUT	동작 설명		
print_int	\$v0 = 1	\$a0 = integer to print	prints \$a0 to standard output		
print_float	\$v0 = 2	\$f12 = float to print	prints \$f12 to standard output		
print_double	\$v0 = 3	\$f12 = double to print	prints \$f12 to standard output		
print_string	\$v0 = 4	\$a0 = address of first character	prints a character string to standard output		
read_int	\$v0 = 5		integer read from standard input placed in \$v0		
read_float	\$v0 = 6		float read from standard input placed in \$f0		
read_double	\$v0 = 7		double read from standard input placed in \$f0		
read_string	\$v0 = 8	\$a0 = address to place string, \$a1 = max string length	reads standard input into address in \$a0		
sbrk	\$v0 = 9	\$a0 = number of bytes required	\$v0= address of allocated memory Allocates memory from the heap		
exit	\$v0 = 10				
print_char	\$v0 = 11	\$a0 = character (low 8 bits)			
read_char	\$v0 = 12		\$v0 = character (no line feed) echoed		
file_open	\$v0 = 13	\$a0 = full path (zero terminated string with no line feed),	\$v0 = file descriptor		

# Department of Electrical and Computer Engineering Seoul National University

		\$a1 = flags, \$a2 = UNIX octal file	
		mode (0644 for rw-rr)	
		\$a0 = file descriptor,	\$v0 = amount of data in buffer
file_read	\$v0 = 14	\$a1 = buffer address,	from file
		\$a2 = amount to read in bytes	(-1 = error, 0 = end of file)
		\$a0 = file descriptor,	v0 = amount of data in buffer to
file_write	\$v0 = 15	\$a1 = buffer address,	file
		\$a2 = amount to write in bytes	(-1 = error, 0 = end of file)
file_close	\$v0 = 16	\$a0 = file descriptor	

### Table 3 SPIM instruction

Instruction	Meaning
li \$t0, 44	register \$t1의 값을 register \$t0에 복사한다.
	실제로 MIPS ISA에 존재하는 instruction이 아니므로 컴파일러가 자동으
	로 addi \$t0, \$0, 44 등으로 바꾼다.
la \$v0, L1	'L1:'이라는 label이 있을 때, 그 label이 있는 주소를 v0에 넣는다.
move \$t0, \$t1	register \$t1의 값을 register \$t0에 복사한다.
	컴파일러가 자동으로 addu \$t0, \$0, \$t1 등으로 바꾼다.
lui \$t0, 0x4355	Set the upper 16 bits or register \$t0 to the
	value 0x4355.

# **III. Requirements**

- 1. SPIM(MIPS ISS)의 사용 법 익히기
- 2. I-type, R-type, J-type 각각의 Instruction에 대한 format과 기능 익히기
- 3. assembly 코딩

보고서에 첨부해야 할 구체적인 내용은 IV. Implementation Details의 각 항목 마지막에 Requirement #1, #2, #3으로 표시하였다.

# **IV. Implementation Details**

## 1. SPIM simulation

앞서 살펴 본 Example1을 SPIM을 통해 시뮬레이션 해 보자.

Department of Electrical and Computer Engineering Seoul National University

🕼 example1.txt (~\Desktop) - GVIM	
파일(E) 편집(E) 도구(T) 문법(S) 버퍼(	B) 창(W) 도움말(H)
486496 * * *	ዄ ፟፟፟፟፟ 🔁 📥 ፟፟ 🙏 🕆 🛍 💶 📍 ኈ
1 .text	# 여기서부터 code가 시작한다는 directive
2 main:	# 'main' 이라는 label
3 1i \$v0, 4	# read number into \$v0
4 la \$a0, str1	# str1 label 주소로 \$a0에 저장
5 syscall	# str1 print
6 li \$v0, 5	# read number into \$v0
7 syscall	<pre># make the syscall read_int</pre>
8 move \$t0, \$v0	# move the number read into \$t0 ≡
9 11 \$v0, 4	
10 1a Şaŭ, str2	H stud suist
	# strz print
12 11 300, 5	# reau second number into \$90
	# make the system read_int
15 add \$t2, \$t8, \$t1	# Move the number read Into Sti
16 1i Su0. 4	
17 1a \$a0, str3	
18 syscall	# str3 print
19 move \$a0, \$t2	# move the number to print into \$a0
20 li \$v0, 1	# load syscall print_int into \$v0
21 syscall	# print \$a0 to console
22 li \$v0, 10	# syscall code 10 is for exit
23 syscall	
24	
25 .data	
26 str1 : .asciiz "Input 1 :"	
27 str2 : .asc11z "Input 2 :"	
28 Str3 : .asc112 "Kesult :"	28,26 모두

메모장이나 VIM 등 텍스트편집툴에 example1의 assemply code를 복사하고 저장한다.

Copen Assembly Code		free fast frequent (Streement) . (Stree		E	x
🚱 🗨 ♥ ■ 바탕 화면 🕨		1 March 1 Marc	▼ ♣ 바탕 화면 검색		٩
구성 ▼ 새 폴더			₩= <b>▼</b>		(?)
☆ 즐겨찾기 🔶	파일 폴더	파일 폴더			^
▶ Dropbox ▶ 다운로드	mips simulator 파일 폴더	Model Checking 파일 폴더			
💻 바탕 화면 🗐 최근 위치	ModelSim 파일 폴더	SPIM 파일 폴더			
➡ 라이브러리 ➡ 문서 ➡ 비디오	survey 파일 폴더	Vísio 파일 풀더			
🛋 사진 🎝 음악	연구실문서_temp 파일 폴더	이혁재교수님 파일 폴더			
🖳 컴퓨터 🚢 로컬 디스크 (C:)	종규형 HEVC 파일 폴더	컴조_교안만들기 파일 폴더			Ш
급 로컬 디스크 (D:) 급 Modesty_storage Transcend (F:) 조	랍미팅 바로 가기 827바이트				-
파일 이	름(N): ex2		<ul> <li>Assembly (*.s *.asm)</li> <li>Assembly (*.s *.asm)</li> <li>Text files (*.txt)</li> </ul>		

SPIM의 File 메뉴 → LoadFile 을 클릭하여 파일 이름 옆의 종류를 Text files(\*.txt)로 바꾸고 저장 한 example1.txt를 불러온다.

🖳 QtSpim			onu	
<u>File</u> Simulator <u>R</u> egisters <u>T</u> ext S	Segment <u>D</u> ata Seg <mark>ment <u>W</u>i</mark>	ndow Help	lory	
📴 🛃 🗔 💷 🏢	🕨 🖬 🖃 🕷	x00400000 부터	시작됨	
' Regs 🛛 Int Regs [16] 🔍 🕨	Data Te:	(t		
Int Regs [16] 🗗 🛪	Text			분러 오 example1 코 ⊑ ×
PC = 0 ^		User Text Sec	ment [00400000]	[00440000]
EPC = 0	[00400000] 8fa40000	lw \$4, O( <mark>\$</mark> 29)	; 183: lw \$a(	<sup>♡ (\$sp</sup> ) <sup>#</sup> 증'르 쪼에 신제 이려하 ㅋㄷ 외쪼에 버
Cause = 0	[00400004] 27a50004	addiu \$5, \$29, 4	; 184: addiu	<sub>\$a1 \$sp 4</sub> 곶 <sub>a</sub> 프gy가에 같게 ㅂㄱ한 ㅗㅗ, 건ㄱ에 ᆫ
BadVAddr = 0	[00400008] 24a60004	addiu \$6, \$5, 4	; 185: addiu	
Status = 3000ff10	[0040000c] 00041080	sll \$2, \$4, 2	; 186: sll \$t	70 Sao 2 역안 Instruction에 들어가고 맨 앞에는
HT = 0	[00400010] 00c23021	addu \$6, \$6, \$2	; 187: addu \$	
LO = 0	[00400014] 0c100009	jal 0x00400024 [main]	; 188: jal ma	*** 수소가 있다
		nop	; 189: nop	
R0 [r0] = 0	[00400012] 3402000a	ori \$2, \$0, 10	; 191: 11 \$00	
R1 [at] = 0	1004000201 00000002	svscall	: 192: SVSCA	( f word number into Sur
R2 [v0] = 0	[00400024] 34020004	lni \$1 4097 [etr1]	, J. II 900,	stri f stri label Áčkó f SaO: : Áúlá
R3 [v1] = 0	[00400020] 30011001	ori $\$4$ , $\$1$ , 0 [str1]	, 4. 14 940,	Stil + Stil label A040-1 \$2007 ALAd
R4 [a0] = 1	[00400030] 000000c	syscall	: 5: syscall	# str1 print
R5 [a1] = /IIII9aC R6 [a2] = 7ffff9b4	[00400034] 34020005	ori \$2, \$0, 5	: 6: 1i \$v0.	5 # read number into \$v0
R7 [a3] = 0	[00400038] 000000c	svscall	; 7: svscall	# make the syscall read int
R8 [t0] = 0	[0040003c] 00024021	addu \$8, \$0, \$2	; 8: move \$t(	0, \$v0 # move the number read into \$t0
R9 $[t1] = 0$	[00400040] 34020004	ori \$2, \$0, 4	; 9: 1i \$v0,	4
R10 [t2] = 0	[00400044] 3c011001	lui \$1, 4097 [str2]	; 10: la \$a0,	, str2
R11 [t3] = 0	[00400048] 3424000a	ori \$4, \$1, 10 [str2]		
R12 [t4] = 0	[0040004c] 000000c	syscall	; 11: syscall	l # str2 print
R13 [t5] = 0	[00400050] 34020005	ori \$2, \$0, 5	; 12: li \$v0,	, 5 # read second number into \$v0
R14 [t6] = 0	[00400054] 000000c	syscall	; 13: syscall	<pre># make the syscall read_int</pre>
R15 [t7] = 0	[00400058] 00024821	addu \$9, \$0, \$2	; 14: move \$t	t1, \$v0 # move the number read into \$t1
R16 [S0] = 0	[0040005c] 01095020	add \$10, \$8, \$9	; 15: add \$t2	2, \$t0, \$t1
R17 [S1] = 0 R18 [s2] = 0	[00400060] 34020004	ori \$2, \$0, 4	; 16: li \$v0,	. 4
R19 [s3] = 0	[00400064] 3c011001	lui \$1, 4097 [str3]	; 17: la \$a0,	. str3
R20 [s4] = 0	[00400068] 34240014	ori \$4, \$1, 20 [str3]		
R21 [s5] = 0	[0040006c] 000000c	syscall	; 18: syscall	1 # str3 print
R22 [s6] = 0	[00400070] 000a2021	addu \$4, \$0, \$10	; 19: move \$a	a0, \$t2 # move the number to print into \$a0
R23 [s7] = 0		ori \$2, \$0, 1	; 20: 11 \$v0,	, 1 # load syscall print_int into \$v0
R24 [t8] = 0	[00400078] 000000c	syscall	; 21: syscall	# print \$a0 to console
R25 [t9] = 0	[00400076] 3402000a	ori \$2, \$0, 10	; 22: 11 \$00,	, 10 # syscall code 10 is for exit
$R_{26}$ [k0] = 0	[00400080] 0000008	syscall	; 23: syscall	
$\mathbf{R}\mathbf{Z} \mathbf{i} [\mathbf{K}\mathbf{I}] = 0 \qquad \mathbf{v}$				·
SPIM Version 9.1.4 of September Copyright 1990-2010, James R. La All Rights Reserved, SPIM is distributed under a BSD See the file README for a full co	4, 2011 arus, license, opyright notice,			

파일을 열어보면 기대와 달리 우리가 불러온 assemply code만 나오는 것이 아니라 복잡하게 다른 instruction들이 보인다. 이는 OS가 없이도 console 등을 사용하여 가상으로 machine code를 테스트 하기 위한 내용이므로 일일이 이해할 필 요는 없다.

또한 메모리 영역이 자동으로 할당된다. Figure5의 Memory map에 나타난 것 처럼 .text 로 표시한 0x00400000 주소부터 코드가 저장되 고, stack은 0x7ffffff에서부터 아래로 자라는 형식 이 된다.



Figure 2 SPIM memory map

Department of Electrical and Computer Engineering Seoul National University

🖳 QtSpim	1		- and - 1 100	-			Data 영역	Mer	nory			3 <mark>X</mark>
File Sin	nulator Registers	Text S	egment Data Sec	iment Windov	v Help							
1				_	·	0x1	0000000	무터	시작됨			
🖾 🖻				=] 🔍							1	
Regs	Int Regs [16]	•	Data	Text								
Int Regs [	16]	ð ×	Data									đΧ
PC	= 0	*	User data se	gment [1000	0000][10	040000]						
EPC	= 0		[10000000]	[1003ffff]	00000000							
Cause	= 0											
BadVAdd	ir = 0											
Status	= 3000ff10		User Stack [	7ffff984]	[80000000]							=
	- 0		[/IIII984]	00000003	/IIIIa3e	/IIIIA39	76666620	2.2.2	. > 9		•	
HI IO	= 0		[/IIII990]	/IIIIa2D	26666666	/IIIIIel	/IIIIID8	+ • •				
10	- 0		[/IIII9a0]	75555522	75555512	71111140	71111137 Zffffodo	:			. /	
0 170	11 = 0		[7ffff9c0]	7ffffec5	7ffffaga	7ffffee0	7ffffd1b	· · ·				
R1 [at	-1 = 0		[7ffff9d0]	7ffffcdd	7ffffcc2	7ffffc7d	7ffffc6b					
R2 [W]	11 = 0		[7ffff9e0]	7ffffc53	7ffffc38	7ffffc1a	7ffffhd9	· · ·	8			
R3 [v1	1 = 0		[7ffff9f0]	7ffffhc2	7ffffbae	7ffffh9f	7ffffh89					
R4 [al	11 = 3		[7ffffa00]	7ffffb60	7ffffb38	7ffffb00	7ffffaea	200				
R5 [a1	1 = 7ffff988		[7ffffa10]	7ffffad9	7fffabc	7ffffa72	7ffffa60		r			
R6 [a2	21 = 7ffff998	=	[7ffffa20]	00000000	00000000	65000000	706d6178				examp	
R7 fa3	31 = 0		[7ffffa30]	652f656c	732e3278	646f6300	3a430065	1 e / (	e x 2 . s .	c o	de.C:	
R8 ItC	01 = 0		[7ffffa40]	6573552f	4d2f7372	7365646f	442£7974	/ U s (	ers/Mo	de	stv/D	
R9 It1	1 = 0		[7ffffa50]	746b7365	612f706f	6d657373	00796c70	esk:	top/as	s e	mplv.	
R10 [t2	2] = 0		[7ffffa60]	646e6977	433d7269	69575c3a	776f646e	wind	dir=C:	\ W	indow	
R11 [t3	3] = 0		[7ffffa70]	53560073	4f433039	4f544e4d	3d534c4f	s.V:	зэосом	ΝΤ	0 0 L S =	
R12 [t4	1] = 0		[7ffffa80]	505c3a43	72676f72	46206d61	73656c69	c : \ :	Progra	m	Files	
R13 [t5	5] = 0		[7ffffa90]	63694d5c	6f736f72	56207466	61757369	\ M i (	crosof	t	Visua	
R14 [te	5] = 0		[7ffffaa0]	7453206c	6f696475	302e3920	6d6f435c	1 S 1	tudio	9.	0 \ C o m	
R15 [t7	7] = 0		[7ffffab0]	376e6f6d	6f6f545c	005c736c	52455355	mon	$7 \setminus T \circ \circ 1$	s \	. U S E R	
R16 [s0	0] = 0		[7ffffac0]	464f5250	3d454c49	555c3a43	73726573	PROD	F I L E = C	: \	Users	
R17 [s1	L] = 0		[7ffffad0]	646f4d5c	79747365	45535500	4d414e52	\ M o d	desty.	υs	ERNAM	
R18 [s2	2] = 0		[7ffffae0]	6f4d3d45	74736564	53550079	4f445245	E = M	odesty	. U	SERDO	
R19 [s3	3] = 0		[7ffffaf0]	4e49414d	646f4d3d	79747365	0043502d	MAII	N = Mode	s t	у-РС.	
R20 [s4	1] = 0		[7ffffb00]	52544c55	4e4f4d41	4e414c5f	52494447	ULTI	RAMON_	LA	NGDIR	
R21 [s5	5] = 0		[7ffffb10]	5c3a433d	676f7250	206d6172	65606946	= C :	\ Progr	a m	File	
R22 [s6	5] = 0		[7ffffb20]	6c555c73	4d617274	525c6e6f	756f7365	s \ U :	ltraMo	n \	Resou	
R23 [s7	7] = 0		[7ffffb30]	73656372	006f6b5c	3d504d54	555c3a43	rcea	s \ k o . T	ΜP	= C : \ U	
R24 [t8	3] = 0		[7ffffb40]	73726573	646f4d5c	79747365	7070415c	ser:	s \ M o d e	s t	ү∖Арр	
R25 [t9	9] = 0		[7ffffb50]	61746144	636f4c5c	545c6c61	00706d65	Dati	a \ L o c a	1 \	Temp.	
R26 [k0	0] = 0		[7ffffb60]	504d4554	5c3a433d	72657355	6f4d5c73	TEMI	$P = C : \setminus U$	s e	r s \ M o	
R27 [k1	[] = 0	-	[7ffffb70]	74736564	70415c79	74614470	6f4c5c61	des	tу∖Арр	Da	t a \ L o	Ŧ
SPIM Ver	rsion 9,1,4 of Septer	mber	4, 2011									*
All Rights	Reserved.	н, La	iius,									
SPIM is	distributed under a	BSD	license,									
See the f	file README for a f	ull co	pyright notice,									-

위 그림처럼 Data tap을 클릭하고 스크롤을 올려보면 제일 위에 User data segment라고 해서 load, store instruction 수행 시 데이터를 메모리에 저장하고 불러올 때 쓰는 메모리 영역이 바로 여기가 된다.

컴퓨터조직론

Fall 2012

### Department of Electrical and Computer Engineering

Seoul National University

🖳 QtSpin	1				simulation co	ntrol menu	
File Sin	nulator Registers	Tevt	eriment Data Se	ament Wir	dow Help		
		<u>_ione</u> c	eginene <u>D</u> ata se	griterit <u>iti</u> n	<u>1100 11</u> 010		
📔 📂 📩	9 🖬 🖪 🛛	51 III	🕨 🖬 📾	Ξ; 🕐			
Reas	Int Regs [16]		Data	Text			
Int Beas [	161	₽ X	Text				₽ X
interiogo p	-		TON				
PC	= 0	- Â		E- 40000	User Text Segmen	t [00400000][004400	
Cance	= 0		[00400000] 8	1a40000	1w \$4, U(\$29)	; 183: 1W \$a0 0(\$sp	) # argc
BadVAdo	ir = 0		[00400004] 2	/a50004	addiu \$5, \$29, 4	; 184: addiu sai ss	p 4 # argv
Status	= 3000ff10		[00400008] 2	4460004	addiu \$6, \$5, 4	; 185: addil \$42 \$4	1 4 # envp
			[00400002] 0	0041080	sii \$2, \$4, 2	; 188: SII \$VU \$40	2
HI	= 0		[00400010] 0	-100000	addu \$6, \$6, \$2	; 187; addu şaz şaz	\$00
LO	= 0		[00400014] 0	000009	jai uxuu4uuu24 [main]	; 188: jai main	
			[00400018] 0	402000-	nop	; 189; nop	=
R0 [r0	0] = 0		[00400016] 3	402000a	ori \$2, \$0, 10	; 191: 11 \$40 10	
R1 [at	:] = 0		[00400020] 0	0000000	syscall	; 192: syscall # sy	scall 10 (exit)
R2 [v0	0] = 0		[00400024] 3	4020005	or1 \$2, \$0, 5	; 3: 11 \$V0, 5 # re	aa number into \$v0
R3 [v1	[] = 0		[00400028] 0	0000000	syscall	; 4: syscall # make	the syscall read_int
R4 [a0	0] = 3		[0040002c] 0	0024021	addu \$8, \$0, \$2	; 5: move \$t0, \$v0	# move the number read into \$t0
R5 [a1	] = 7ffff988		[00400030] 3	4020005	ori \$2, \$0, 5	; 6: 1i \$v0, 5 # re	ad second number into \$v0
R6 [a2	<pre>2] = 7ffff998</pre>	=	[00400034] 0	000000c	syscall	; 7: syscall # make	the syscall read_int
R7 [a3	3] = 0		[00400038] 0	0024821	addu \$9, \$0, \$2	; 8: move \$t1, \$v0	# move the number read into \$t1
R8 [t0	0] = 0		[0040003c] 0	1095020	add \$10, \$8, \$9	; 9: add \$t2, \$t0,	\$t1
R9 [t1	1] = 0		[00400040] 0	00a2021	addu \$4, \$0, \$10	; 10: move \$a0, \$t2	# move the number to print
R10 [t2	2] = 0		into \$a0				
R11 [t3	3] = 0		[00400044] 3	4020001	ori \$2, \$0, 1	; 11: li \$v0, 1 # 1	oad syscall print_int into \$v0
R12 [t4	i] = 0		[00400048] 0	000000c	syscall	; 12: syscall # pri	nt \$a0 to console
R13 [t3	5] = 0		[0040004c] 3	402000a	ori \$2, \$0, 10	; 13: li \$v0, 10 #	syscall code 10 is for exit
R14 [00	b] = 0		[00400050] 0	00000c	syscall	; 14: syscall	
D16 [c]	j = 0						
R10 [30					Kernel Text Segme	nt [80000000][80010	0000]
R18 [s2	21 = 0		[80000180] 0	001d821	addu \$27, \$0, \$1	; 90: move \$k1 \$at	# Save \$at
R19 [83	31 = 0		[80000184] 3	c019000	lui \$1, -28672	; 92: sv \$v0 s1 # N	ot re-entrant and we can't
R20 [s4	11 = 0		trust \$sp				
R21 [s	51 = 0		[80000188] a	c220200	sw \$2, 512(\$1)		
R22 [s6	51 = 0		[8000018c] 3	c019000	lui \$1, -28672	; 93: sw \$a0 s2 # E	ut we need to use these
R23 [s7	1 = 0		registers				
R24 [t8	3] = 0		[80000190] a	c240204	sw \$4, 516(\$1)		
R25 [t9	0 = 0		[80000194] 4	01a6800	mfc0 \$26, \$13	; 95: mfc0 \$k0 \$13	# Cause register
R26 [k0	0] = 0		[80000198] 0	01a2082	srl \$4, \$26, 2	; 96: srl \$a0 \$k0 2	# Extract ExcCode Field
R27 [k1	l] = 0	-	[8000019c] 3	084001f	andi \$4, \$4, 31	; 97: andi \$a0 \$a0	0x1f -
SPIM Ve	rsion 91,4 of Sep	tember	4, 2011				
Copyright	t 1990-2010, Jame	es R. La	arus,				
SPIM in	is neservea, distributed under	a BSD	licansa				
See the t	file README for a	a full co	ovright notice.				*

이제 시뮬레이션을 실행 해 보자. 위 그림의 메뉴를 사용하면 되는데, 재생버튼을 누르면 모든 instruction이 모두 수행되어 버리므로 중간과정을 관찰할 수 없다. 그래서 가장 오른쪽 메뉴인 single step을 클릭하면(또는 단축키 F10) 0x00400000부터 PC값이 4씩 올라가면서 instruction의 위치를 가리키고, 해당 instruction이 수행되면서 변하는 register의 값이라든지 Data 영역 memory의 변화 등을 관찰할 수 있다.

한 번 simulation을 실행하고 나면 register와 메모리 값, PC 등을 모두 초기화하고 다시 시작해 야 하므로 SPIM의 File 메뉴 → Reinitialize and LoadFile 을 클릭하여 다시 example1.txt를 불러 와서 실행한다. 이번에는 Simulator 메뉴 → Run/Continue (단축키 F5)를 클릭하여 실행하고 console창에 숫자입력 <enter>, 숫자입력 <enter> 하면 두 입력한 숫자의 합이 출력된다.



### Figure 3 example1의 출력화면

Requirement #1
첨부한 example2, 3, 4, 5를 위의 example1에서 처럼 SPIM을 통해 수행한 결과를 보고서에 첨부
하라. 또한 모든 instruction에 대한 설명을 주석으로 달아서 보고서에 첨부하라.

### 2. Instruction

ADD	ADDU	ADDI	ADDIU	SUB	SUBU
SLT	SLTU	SLTI	SLTIU	AND	ANDI
OR	ORI	XOR	XORI	NOR	SRL
SRA	SLL	SRLV	SRAV	SLLV	LW
SW	BEQ	BNE	BLTZ	BLEZ	BGTZ
BGEZ	LUI	J	JAL	JR	

위의 35개 instruction은 프로젝트2부터 구현할 MIPS에서 지원하게 될 Instruction 목록이다. 이 instruction들에 대한 datapath를 구현하려면 instruction의 format, 기능 등을 분류하고 정확히 알 고 있어야 한다. 그렇지 않으면 프로젝트 2에서 상당히 고생하게 될 것이다. R-type/I-type/J-type 으로 분류하고 각각을 아래의 예시 에서 처럼 정리한다.

예	시	
R-1	уре	
	1. ADD	
	Description	Adds two registers and stores the result in a register
	Operation	\$d = \$s + \$t
	Syntax	add \$d, \$s, \$t
	Encoding	0000 00ss ssst tttt dddd d000 0010 0000

I-type

# 1. ADDI

Description	Adds a register and a sign-extended immediate value and stores the result			
Description	in a register			
Operation	\$t = \$s + imm			
Syntax	addi \$t, \$s, imm			
Encoding	0010 00ss ssst tttt iiii iiii iiii iiii			

### J-type

Description	Jumps to the calculated address	
Operation	PC = (PC & 0xf000000)   (target << 2)	
Syntax	j target	
Encoding	0000 10ii iiii iiii iiii iiii iiii	

### **Requirement #2**

35개의 instruction에 대해 R-type/I-type/J-type으로 분류하고 위의 양식에 맞춰 그 내용을 기술 하여 보고서에 첨부한다. 내용이 정확하면서 본인이 알아보기 쉽도록 하는 것이 중요하다.

### 3. Assembly coding

지금까지 익힌 SPIM의 사용법과 Instruction에 대한 이해를 바탕으로 직접 assembly coding을 해 본다. 아래의 C코드에 대응하는 assembly 코드를 작성하고 SPIM을 사용하여 테스트 해 본다.

```
#include <stdio.h>
void main() {
    int a[7] = {-36, 20, -27, 15, 1, -62, -41};
    int n = 7;
    int i;
    int npos=0;
    int nneg=0;

    printf("Computer Organization, Fall 2011\n");
    printf("SPIM simulation\n");
    printf("Athor : 본인이름, StudentNum : 학번\n");

    for (i = 0; i < n; i++) {
        if(a[i] > 0)
            npos++;
        else
    }
}
```

```
nneg++;
}
printf("Number of pos : %d\n", npos);
printf("Number of neg : %d\n", nneg);
}
```

# **Requirement #3**

위의 C코드에 상응하는 assembly 코드를 작성하고 주석을 달아 보고서에 첨부한다. 작성한 코드 파일 또한 따로 제출한다.

### **Additional Requirement**

필수는 아니지만 성의 있게 수행하면 가산점을 받을 수 있다. Requirement#2에서 조사한 35개의 instruction을 최대한 많은 수를 활용하여 자유주제로 assembly 코드를 작성 해 본다. 프로그램의 수준은 상관없고 단순히 instruction을 사용한 수가 많을수록 좋은 평가를 받는다.

### V. Submission

제출과제			
1.	보고서(이메일과 hardcopy 모두 제출)		
	1	Requirement#1	
		example2~5 simulation 결과 캡쳐 첨부, 주석 달린 코드 내용을 문서에 첨부	
	2	Requirement#2	
		35개의 instruction 정리한 내용 첨부	
	3	Requirement#3	
		본인이 작성한 Assembly code에 주석 달아서 보고서에 첨부, simulation 결과 첨부	
2.	Req	uirement #3에서 본인이 작성한 assemply code 파일(이메일 제출)	

제출방법		
1. 조교 <b>이메일 <u>modesty@sdgroup.snu.ac.kr</u> 로 보고서, 코드파일을 압축하여 제출.</b>		
2. 보고서 Hardcopy는 각자 출력해서 10월 9일 수업시간 전에 제출		
※ 이메일 제출은 10/8일 월요일 저녁 11시 59분 까지 제출분만 인정(메일 도착 시간 기준)		

# **APPENDIX A Background knowledge**

### 1. Assembly & machine language

C언어로 프로그래밍코드를 작성하고 컴파일하고 실행파일을 만들어 application을 수행하는 일 련의 프로그래밍 경험은 다들 있을 것이다. 여기에는 여러 가지 전제가 있는데, 우선 대부분의 PC에서 사용하고 있는 Processor는 Intel architecture를 사용하고 있다. 즉 ISA(Instruction Set Architecture)가 Intel target이라면 이 ISA에서 지원하는 instruction으로 소스코드가 컴파일 된다. 즉 보통 우리가 사용하고 있는 컴파일러는 정확히 말하면 Inter의 x86 ISA용 instruction을 사용하 여 C코드를 x86 ISA가 알아들을 수 있는 machine language로 번역해 주는 툴 이라고 할 수 있다. 이 컴파일러를 통해 나온 실행파일은 Intel processor에서만 실행 가능하다. 실제 MIPS ISA용 컴파 일러로 C코드를 컴파일 하여 실행하면 잘못 된 비트파일이라는 메시지가 나온다.

다시 말해 어떤 소스코드를 우리가 설계하는 MIPS ISA를 타겟으로 테스트하기 위해서는 첫째로 MIPS ISA를 지원하는 cross-compiler가 필요하고, 둘째로 컴파일 된 machine code를 실행 할 MIPS 프로세서가 필요하다.

gcc는 GNU에서 만든 컴파일러 이다. GNU에 대해서는 각자 자료조사를 해 보시길 바란다. gcc 를 사용한 컴파일과정은 아래와 같다.



Figure 1 gcc 컴파일 과정<sup>1</sup>

실제로 mipsel-linux-gcc 라는 MIPS ISA 타겟의 cross-compiler를 설치하고 컴파일한 결과를 살 펴보자. 아래의 명령을 수행하면 -v 옵션에 의해 컴파일 과정이 화면에 출력되고, --save-temps 옵션에 의해 컴파일 과정에서 발생하는 중간 파일들(test.s, test.i 등)이 모두 저장된다. 실제로 테 스트 해 볼 간단한 c source에 대해 machine language code 생성과정을 수행하면 다음과 같은 결과를 얻을 수 있다.

<sup>1</sup>백창우,『유닉스·리눅스 프로그래밍 필수 유틸리티 : vi, make, gcc, gdb, cvs, rpm』,한빛미디어㈜,1999, p.164.

#### \*명령어 : mipsel-linux-gcc -v --save-temps -o run test.c

ImdestyBungte -/mips\_testj8 mipsel-linux-goc -v --save-tesps -0 run test.c Reading specs from /home/modesty/usr/bin/../lib/goc/mipsel-linux/3.4.4/specs Configured with: ./configure --targetemipsel-linux --prefixe/usr --program-prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --prefixe/usr --prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --prefixe/usr --targetemipsel-linux --vith-headerse/project/toolchain\_mips\_le/kernel-2.6-x/fedora-core-x/linux/include --enable-languagese.c++ Thread model: posix goc version 3.4.4 /home/modesty/usr/bin/../lib/goc/mipsel-linux/3.4.4/col -E -quiet -v -iprefix /home/modesty/usr/bin/../lib/goc/mipsel-linux/3.4.4/ test.c -o test.i ignoring nonexistent directory "/usr/lib/goc/mipsel-linux/3.4.4/include" #include "..." search starts here: /home/modesty/usr/bin/../lib/goc/mipsel-linux/3.4.4/.../././.mipsel-linux/ays-include /home/modesty/usr/bin/../lib/goc/mipsel-linux/3.4.4/.../././.mipsel-linux/ays-include /home/modesty/usr/bin/../lib/goc/mipsel-linux/3.4.4/.ccl -fpreprocessed test.i -quiet -dumpbase test.c -auxbase test -version -o test.s GNU C version 3.4.4 (mipsel-linux/s.4.4/.ccl -fpreprocessed test.i -quiet -dumpbase test.c -auxbase test -version -o test.s GNU C version 3.4.4 (mipsel-linux/s.4.4/.ccl -fpreprocessed test.i -quiet -dumpbase test.c -auxbase test -version -o test.s GNU C version 3.4.4 (mipsel-linux/s.4.4/.ccl -fpreprocessed test.i -quiet -dumpbase test.c -auxbase test -version -o test.s GNU C version 3.4.4 (mipsel-linux/s.4.4/.ccl -fpreprocessed test.i -quiet -dumpbase test.c -auxbase test -version -o test.s GNU C version 3.4.4 (mipsel-linux/s.4.4/.ccl -fpreprocessed test.i -quiet -dumpbase test.c -auxbase test -version -o test.s GNU C version 3.4.4 (mipsel-linux/s.4.4/.ccl -c.frame-hdr =EL -no-mdebug -32 -v -KPIC -o test.o test.s GNU c vers



	1 void main(){
1	2
1 main:	<pre>3 int i = 0;</pre>
2 addiu şsp,şsp,-24	<pre>4 int j = 0;</pre>
3 sw \$fp,16(\$sp)	5
4 move şfp,şsp	<pre>6 for(i = 0; i &lt; 10; i++)</pre>
5 sw \$0,8(\$fp)	7 j += i;
6 sw \$0,12(\$fp)	8 }
7 sw \$0,8(\$fp)	1 test c (c source file)
8 \$L2:	
9 lw \$2,8(\$fp)	2 101001111011110111111111111111101000
10 slt \$2,\$2,10	
11 beq \$2,\$0,\$L1	
12 lw \$3,12(\$fp)	
13 lw \$2,8(\$fp)	5 1010111111000000000000000000000000000
14 addu \$2,\$3,\$2	6 1010111111000000000000000000000000000
15 sw \$2,12(\$fp)	7 1000111111000010000000000000000000
16 lw \$2,8(\$fp)	8 0010100001000010000000000000000000000
17 addiu \$2,\$2,1	9 00010000010000000000000000000000
18 sw \$2,8(\$fp)	10 10001111110000110000000000001100
19 b \$L2	11 100011111100001000000000000000000
20 \$L1:	12 000000001100010000100000100001
21 move \$sp,\$fp	13 10101111110000100000000000001100
22 lw \$fp,16(\$sp)	14 100011111100001000000000000000000
23 addiu \$sp,\$sp,24	15 00100100010000100000000000000000
24 j \$31	16 101011111100001000000000000000000
-	17 0001000000000001111111111110101
	18 00000011110000001110100000100001
	19 1000111110111110000000000000000000
	20 00100111101111010000000000011000
	21 000000111110000000000000000000000000
2 test.s (assembly language code)	③ machine language code

Figure 3 MIPS machine code 생성 과정

#### 2. ISS(Instruction Set Simulator)

소스코드를 우리가 설계하는 MIPS ISA를 타겟으로 테스트하기 위해 첫째로 cross-compiler, 둘 째로 컴파일 된 machine code를 실행 할 MIPS 프로세서가 필요하다고 했다. 그런데 우리는 지금 MIPS 프로세서가 없기 때문에 MIPS instruction으로 컴파일 된 machine code가 있어도 실행을 해 볼 수가 없다. 이럴 때 ISS가 필요한 것이다. 실제로는 프로세서를 설계할 때 먼저 ISS를 설계하는 것이 순서이다. 왜냐하면 하드웨어로 프로세서를 설계했을 때 이 것이 제대로 동작하는지 확 인하기 위해 비교해야 할 대상이 필요한데 바로 ISS가 비교 대상이 된다. 게다가 Verilog 시뮬레 이터를 이용하여 회로의 동작을 확인하는 것은 속도가 너무도 느리다. 그래서 ISS는 주로 HDL(Hardware Description Language)가 아닌 C, C++ 등의 상위 언어로 모델링하는 것이 보통이 다. 아래 내용은 IDEC newsletter에 KAIST 배영돈 박사님(donny.ics.kaist.ac.kr)께서 연재하신 '마이 크로프로세서 설계 무작정 따라하기'에 실린 내용의 일부이다.



Figure 4 Microprocessor 개발 환경

C코드를 설계하고자 하는 processor 타겟으로 컴파일하여 최종 object code까지 만들어지면 이 것을 설계한 Microprocessor와 C언어 등으로 모델링 된 ISS에 동시에 넣어 두 결과를 비교함으로 써 설계한 Microprocessor가 제대로 동작하는지 검증할 수 있다.

이번 프로젝트에서는 C언어 등을 사용하여 ISS를 설계하지는 않고 MIPS용으로 많이 사용되는 SPIM이라는 ISS를 사용하여 Instruction을 test하고 assemply coding도 직접 해 본다.