Project 2. MIPS Simulator

Due 23:59, April 13th, 2025

TA: Juhyun Lee, Hyunmin Choi

1. Introduction

In this project, you will build a **simulator for a subset of the MIPS instruction set**. The simulator loads a MIPS binary into a simulated memory then executes the instructions. Instruction execution will change the state of registers and memory of the simulated MIPS system. **Please read this document and README.md file provided in the repository carefully before you start.**

****If you have any questions related to the project, please ask them in the project_2 folder on Piazza.**

2. GitLab Repository

Basically, you just need to repeat the same procedure of forking and cloning the repository explained in the document for Project 1. The only difference is that you should start from the TA's repository of **Project 2** (https://cs311.kaist.ac.kr/handout/project2).

Please check prior notifications available in Piazza board and KLMS, for more information about getting started with GitLab.

3. Simulator Requirements

For a given input MIPS binary (**the same format of the output binary file from the MIPS assembler built in Project 1**), the simulator must simulate the behaviors of MIPS machine based on the MIPS ISA.

States
The simulator must have the system states, which consist of general purpose registers (GPR; R0-R31), program counter (PC), and memory. The registers (GPRs, PC) and the memory must be created and initialized when a simulation begins.

Loading an input binary
For a given input binary, the loader must identify the text and data section sizes. The text section must be loaded into the simulated memory from the address 0x400000. The data section must be loaded to the simulated memory from the address 0x10000000. In this project, you only need to implement a simple loader which does not create the stack region.

Initial states
* PC: The initial value of PC is 0x400000.
* Registers: All values of register 0 to 31 are set to zero.
* Memory: You may assume all initial values are zero, except for the part of the loaded text and data sections.

Instruction execution
With a current PC, 4-byte (an instruction) is read from the memory. The simulator must parse (decode) the binary instruction and identify what the instruction is and what the operands are. Then, based on the operations defined in MIPS ISA, the simulator must accurately simulate the executions, which will update PC, register, or memory.

Termination
The simulator must stop after executing the given number of instructions (specified in the option for simulator).

Supported Instruction Set (Same as Project 1, more information in Appendix)

ADDIU	ADDU	AND	ANDI	BEQ	BNE	J
JAL*	JR	LUI	LW	LA*	NOR	OR
ORI	SLTIU	SLTU	SLL	SRL	SW	SUBU

- **Delay Slot (Important!!)**
  In project 1, first operation of 'JAL' instruction was described as R31 = (current PC) + 8, due to the delay slot. However, we are going to **ignore the 'delay slot'** for this project.
  Therefore, from project 2, that part was changed into **R31 = (current PC) + 4**.

- **Pseudo-instruction**
  'LA' instruction is a pseudo instruction that can be converted to one or two assembly instructions. Please read Project 1 specification for more information.

4. Simulator Options and Output

Basic command
$ ./cs311sim [-m addr1:addr2] [-d] [-n num_instr] <input file>

Options
-m : Print the memory content from addr1 to addr2
-d : Print the register file content for each instruction execution. Print the memory content together if –m option is specified.
-n : number of instructions simulated

The default output is the PC and register file content after the completion of the given number of instructions. If –m option is specified, the memory content from addr1 to addr2 must be printed too.

If –d option is set, the register (and memory dump, if –m is enabled) must be printed for every instruction execution.

Formatting Output
PC and register content must be printed in addition to the optional memory content.
You should print the output with standard output.

1. If you type the command line as below, the output file should show only PC and register values like **Figure 1**.
   $ ./cs311sim –n 0 input.o

2. If you type the command line as below, the output file should show memory contents of specific memory region, PC and register values like **Figure 2.**
   $ ./cs311sim –m 0x400000:0x400010 –n 0 input.o

3. The functions for printing the memory and register values are available in the util.c, util.h file.

```
Simulating for 0 cycles...

Current register values :
-------------------------------------
PC: 0x00400000
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x00000000
R3: 0x00000000
R4: 0x00000000
R5: 0x00000000
R6: 0x00000000
R7: 0x00000000
R8: 0x00000000
R9: 0x00000000
R10: 0x00000000
R11: 0x00000000
R12: 0x00000000
R13: 0x00000000
R14: 0x00000000
R15: 0x00000000
R16: 0x00000000
R17: 0x00000000
R18: 0x00000000
R19: 0x00000000
R20: 0x00000000
R21: 0x00000000
R22: 0x00000000
R23: 0x00000000
R24: 0x00000000
R25: 0x00000000
R26: 0x00000000
R27: 0x00000000
R28: 0x00000000
R29: 0x00000000
R30: 0x00000000
R31: 0x00000000
```

Figure 1. Default Output

```
Simulating for 0 cycles...

Current register values :
-------------------------------------
PC: 0x00400000
Registers:
R0: 0x00000000
R1: 0x00000000
R2: 0x00000000
R3: 0x00000000
R4: 0x00000000
R5: 0x00000000
R6: 0x00000000
R7: 0x00000000
R8: 0x00000000
R9: 0x00000000
R10: 0x00000000
R11: 0x00000000
R12: 0x00000000
R13: 0x00000000
R14: 0x00000000
R15: 0x00000000
R16: 0x00000000
R17: 0x00000000
R18: 0x00000000
R19: 0x00000000
R20: 0x00000000
R21: 0x00000000
R22: 0x00000000
R23: 0x00000000
R24: 0x00000000
R25: 0x00000000
R26: 0x00000000
R27: 0x00000000
R28: 0x00000000
R29: 0x00000000
R30: 0x00000000
R31: 0x00000000

Memory content [0x00400000..0x00400010] :
-------------------------------------
0x00400000: 0x02208824
0x00400004: 0x02409024
0x00400008: 0x3c081000
0x0040000c: 0x3c091000
0x00400010: 0x35290004
```

Figure 2. Output with Option ‘-m’

5. Grading Policy

Submissions will be graded based on the 10 examples**: 6 given examples in 'sample_input' directory, and 4 hidden cases.** Hidden cases have similar complexity with respect to given examples.

To get score, your simulator should print the exactly same output as the reference solution. You can evaluate your code within given examples by comparing your output with files in 'sample_output' directory. **You may check the correctness of your code by executing** `make test` **at your working directory of this project**.

If there are any differences (including whitespaces) it will print the differences as below (here, there are two different lines corresponding to register R4 and R31). If there is no difference for an example, it will print "Test seems correct".

```
Testing example01
        Test seems correct

Testing example02
        Test seems correct

Testing example03
        Test seems correct

Testing example04
        Test seems correct

Testing example05
        Test seems correct

Testing fact
--- sample_output/fact  2019-09-26 21:29:59.290040281 +0900
+++ -   2019-09-26 21:30:04.128396077 +0900
@@ -8,7 +8,7 @@
 R1: 0x00000000
 R2: 0x00000001
 R3: 0x00000000
-R4: 0xfff1ffec
+R4: 0xffffffec
 R5: 0x00000000
 R6: 0x00000000
 R7: 0x00000000
@@ -35,5 +35,5 @@
 R28: 0x00000000
 R29: 0x00000000
 R30: 0x00000000
-R31: 0x0040201c
+R31: 0x0040001c

        Results not identical, check the diff output
```

Please make sure your outputs for given examples are **identical to the files in the sample_output directory, without any redundant prints. Every single character of the output must be identical** to the given sample output. Otherwise, you will receive **0 score** for the example.

Note that **simply hard-coding outputs for given examples would lead you to 0 score** for this project.

6. Submission (Important!!)

Make sure your code works well on our class Linux server.
It is highly recommended to work on the class server, since your project will be graded on the same environment as those servers.

Add the 'submit' tag to your final commit and push your work to the gitlab server.
The following commands are the flow you should take to submit your work:

```
git add .
git commit -m “your_custom_message”
git tag -a submit -m “your_custom_message”
git push origin main :submit
```

If you have modified the code after submission or submitted it incorrectly, you need to delete the tag and create it again.
Type the following command in your working directory to delete tag:

```
git tag -d submit
git push origin main :submit
```

And then, add the ‘submit’ tag again.

If there is no “submit” tag, your work will not be graded. Please do not forget to submit your work with the tag.

Please make sure you push your work before the deadline. If you do not “push” your work, we won’t be able to see your work so that your work will be marked as unsubmitted.

For more information about submission procedure, please refer to the specification of project 1 “Submission”.

7. Late Policy & Plagiarism Penalty (Important!!)

You will lose **30%** of your score on the **first day** (April 14th 0:00~23:59). We will **not accept** any works that are submitted after then.

Be aware of plagiarism! Although it is encouraged to discuss with others and refer to extra materials, **copying other students’ or opened code is strictly banned: Not only for main routine functions, but also helper functions.**

TAs will compare your source code with open-source codes and other students’ code. If you are caught, you will receive a serious penalty for plagiarism as announced in the first lecture of this course.

If you have any requests or questions regarding administrative issues (such as late submission due to an unfortunate accident, GitLab is not working) please send an e-mail to TAs.

8. Tips

Modifying your repository via http may cause some problems.

You may use C++ instead of C if you want. Just make sure your '`make test`' command works properly.

Please, do not modify any files other than parse.c, run.c, and Makefile.
(Simply modifying filename extension from .c to .cpp without modification on contents is acceptable however.)
Only the three files — parse.c, run.c, and Makefile(for those using C++) — submitted in the submit tag will be used for grading. To ensure the code runs correctly during evaluation, do not modify any other files.

Please start as early as possible.

Please **read this document, README.md, and given skeleton code carefully before you start.**

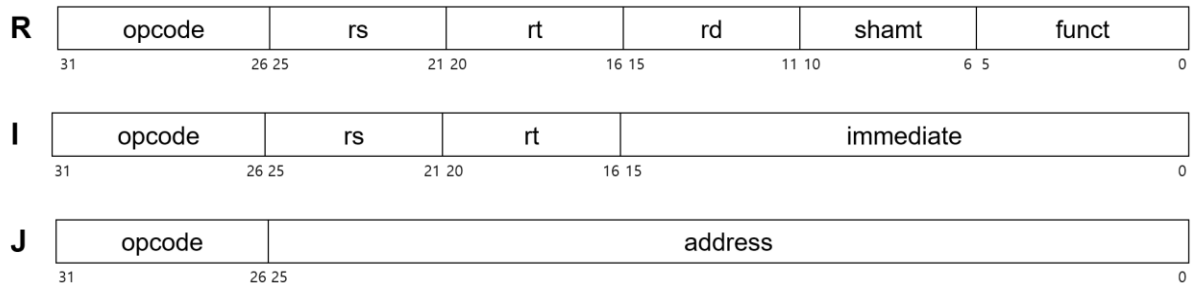
Appendix. Detailed Information about Instruction Set in Project 2

1. Core Instruction Set

NAME	MNE MONIC	FOR MAT	OPERATION (in Verilog)	OPCODE	FUNCT
Add Immediate Unsigned	ADDIU	I	R[rt] = R[rs] + SignExtImm (1)	0x9	-
Add Unsigned	ADDU	R	R[rd] = R[rs] + R[rt]	0x0	0x21
And	AND	R	R[rd] = R[rs] & R[rt]	0x0	0x24
And Immediate	ANDI	I	R[rt] = R[rs] & ZeroExtImm (2)	0xc	-
Branch On Equal	BEQ	I	if (R[rs] == R[rt]) PC = PC+4+BranchAddr (3)	0x4	-
Branch On Not Equal	BNE	I	if (R[rs] != R[rt]) PC = PC+4+BranchAddr (3)	0x5	-
Jump	J	J	PC = JumpAddr (4)	0x2	-
Jump And Link	JAL	J	R[31] = PC + 4; PC = JumpAddr	0x3	-
Jump Register	JR	R	PC = R[rs]	0x0	0x8
Load Upper Immediate	LUI	I	R[rt] = {imm, 16'b0}	0xf	-
Load Word	LW	I	R[rt] = M[R[rs] + SignExtImm (1)	0x23	-
Nor	NOR	R	R[rd] = ~(R[rs] | R[rt])	0x0	0x27
Or	OR	R	R[rd] = R[rs] | R[rt]	0x0	0x25
Or Immediate	ORI	I	R[rt] = R[rs] | ZeroExtImm (2)	0xd	-
Set Less Than Immediate Unsigned	SLTIU	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0 (1) (5)	0xb	-
Set Less Than Unsigned	SLTU	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0 (5)	0x0	0x2b
Shift Left Logical	SLL	R	R[rd] = R[rt] << shamt	0x0	0x00
Shift Right Logical	SRL	R	R[rd] = R[rt] >> shamt	0x0	0x02
Store Word	SW	I	M[R[rs] + SignExtImm] = R[rt] (1)	0x2b	-
Subtract Unsigned	SUBU	R	R[rd] = R[rs] – R[rt]	0x0	0x23

(1) SignExtImm = {16{immediate[15]}, immediate}
(2) ZeroExtImm = {16{1b'0}, immediate}
(3) BranchAddr = {14{immediate[15]}, immediate, 2'b0}
(4) JumpAddr = {PC+4[31:28], address, 2'b0}
(5) Operands considered unsigned numbers (vs. 2's complement)

2. Basic Instruction Formats



OPCODES, BASE CONVERSION, ASCII SYMBOLS

③

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(1)	sll	add.*f*	00 0000	0	0	NUL	64	40	@
		sub.*f*	00 0001	1	1	SOH	65	41	A
j	srl	mul.*f*	00 0010	2	2	STX	66	42	B
jal	sra	div.*f*	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.*f*	00 0100	4	4	EOT	68	44	D
bne		abs.*f*	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.*f*	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.*f*	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.*f*	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.*f*	00 1101	13	d	CR	77	4d	M
xori		ceil.w.*f*	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.*f*	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.*f*	01 0010	18	12	DC2	82	52	R
	mtlo	movn.*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.*f*	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.*f*	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.*f*	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.*f*	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.*f*	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.*f*	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.*f*	11 0011	51	33	3	115	73	s
	teq	c.olt.*f*	11 0100	52	34	4	116	74	t
ldc1		c.ult.*f*	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.*f*	11 0110	54	36	6	118	76	v
		c.ule.*f*	11 0111	55	37	7	119	77	w
sc		c.sf.*f*	11 1000	56	38	8	120	78	x
swc1		c.ngle.*f*	11 1001	57	39	9	121	79	y
swc2		c.seq.*f*	11 1010	58	3a	:	122	7a	z
		c.ngl.*f*	11 1011	59	3b	;	123	7b	{
		c.lt.*f*	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.*f*	11 1101	61	3d	=	125	7d	}
sdc2		c.le.*f*	11 1110	62	3e	>	126	7e	~
		c.ngt.*f*	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

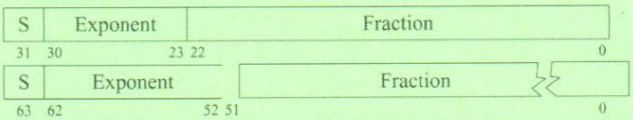
(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127, Double Precision Bias = 1023.

IEEE Single Precision and Double Precision Formats:



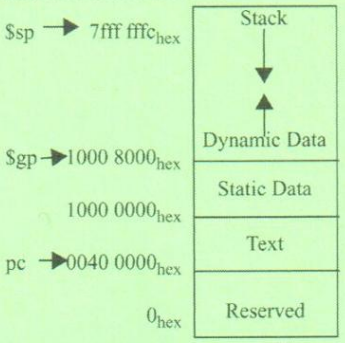
④

IEEE 754 Symbols

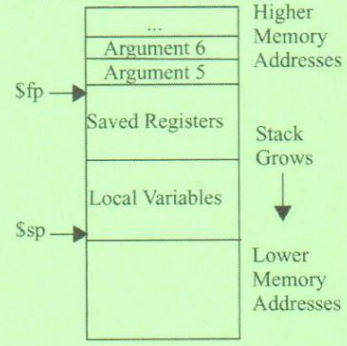
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

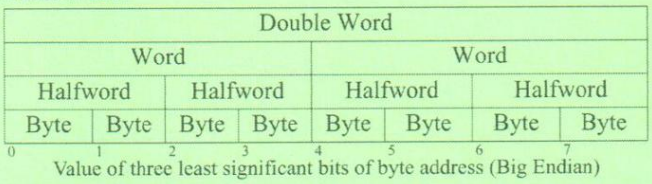
MEMORY ALLOCATION



STACK FRAME

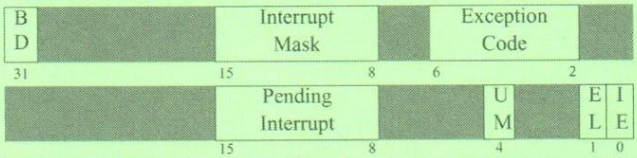


DATA ALIGNMENT



Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES

	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL
10^3	Kilo-	K	2^{10}	Kibi-	Ki	10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^6	Mega-	M	2^{20}	Mebi-	Mi	10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^9	Giga-	G	2^{30}	Gibi-	Gi	10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti	10^{24}	Yotta-	Y	2^{80}	Yobi-	Yi