Assembly Language Project

Your job is to write an assembler much like the SPIM assembler.

Your assembler will write the text and data segments. Text segment comes first and will be indicated with the line containing just [00400024]. Succeeding lines will be the code that is loaded there. Eight 32 bit words per line (in hex).

The data segment will start with the line containing just [10010000], followed by data as with the text segment.

My best advice is to write assembly code in MIPS. Figure out what you think it should generate, then run it through QTSPIM or SPIM and see what it actually generates. I have decoded all the pseudoinstructions into the assembly code they represent. Feel free to post clarifying questions on piazza. I will make this document as definitive as possible and reasonable.

Literals

There are several kinds of literals that can be used depending on the command. Single character escape sequences with '\' only work with 'n', 't', and '"'. Octal escape sequences work with values between '\000' and '\077'

strings	"this is a String"
characters	'A'
integers	1, -2, 0x42
floats	3.14159

Addresses

An identifier, followed by a colon, can appear at the front of any line. The assembler will associate this label with the address to where the instruction is assembled. It may appear with branch and jump statements.

Dot Commands

.ascii str	Store the string str in memory, but do not null-terminate it.
.asciiz str	Loads the ASCII characters and null terminator at the next available locations
.word w1,..., wn	Store the n 32-bit quantities in successive memory words.
.space n	Allocate n bytes of space in the current data segment
.byte b1,..., bn	Store the n values in successive bytes of memory.
.float f1,..., fn	Store the n floating-point single precision numbers in successive memory locations.
.double d1,..., dn	Store the n floating-point double precision numbers into memory.
.data	Future data gets assembled to the memory area, starting with 10010000 hex
.end	This has to be the last instruction
.globl	This statement is ignored
.text	Future instructions get assembled to the memory area, starting with 400024 hex

Comments

Ignore all characters after and including the '#' character. Ignore blank lines.

Assembly Instructions

BTW The sequence of fields:
R format is opcode(6), rs(5), rt(5), rd(5), shamt(5), func(6) (shamt=0 except for shift instructions)
I format is opcode(6), rs(5), rt(5), imm(16)
J format is opcode(6), address(26)

BTW The register numbers are $zero(0), $at(1), $v0-v1(2-3), $a0-a3(4-7), $t0-t7(8-15), $s0-s7(16-23), $t8-t9(24-25), $k0-k1(26-27), $gp(28), $sp(29), $fp(30), $ra(31)

instruction		format/ opcode(hex)/ func(hex)	pseudo RTL
R format with registers rd, rs, and rt (shamt=0)			
movz	rd, rs, rt	R/00/0A	Move conditional zero
movn	rd, rs, rt	R/00/0B	Move conditional not zero
add	rd, rs, rt	R/00/20	Addition (with overflow) (rs + rt -> rd)
addu	rd, rs, rt	R/00/21	Addition (without overflow) (rs + rt -> rd)
sub	rd, rs, rt	R/00/22	Subtract (without overflow) (rs - rt -> rd)
subu	rd, rs, rt	R/00/23	Subtract (with overflow) (rs - rt -> rd)
and	rd, rs, rt	R/00/24	AND (rs & rt -> rd)
or	rd, rs, rt	R/00/25	OR (rs | rt -> rd)
xor	rd, rs, rt	R/00/26	Exclusive OR (rs xor rt -> rd)
nor	rd, rs, rt	R/00/27	NOR (rs nor rt -> rd)
slt	rd, rs, rt	R/00/2A	Set less than (rs<rt -> rd)
sltu	rd, rs, rt	R/00/2B	Set less than unsigned (rs<rt -> rd)
mul	rd, rs, rt	R/1C/02	Multiply (without overflow) (rs * rt -> rd)
R format with registers rd, rt, and rs (shamt=0)			
sllv	rd, rt, rs	R/00/04	Shift left logical variable
srlv	rd, rt, rs	R/00/06	Shift right logical variable
srav	rd, rt, rs	R/00/07	Shift right arithmetic variable
R format with registers rd and rt, and shamt (rs=0)			
sll	rd, rt, shamt(0..31)	R/00/00	Shift left logical (rt{31-shamt..0}->rd{31..shamt}, 0->rd{shamt-1..0})
srl	rd, rt, shamt(0..31)	R/00/02	Shift right logical (0->rd{31..(31-shamt+1)}, rt{31..shamt}->rd{31-shamt..0})
sra	rd, rt, shamt(0..31)	R/00/03	Shift right arithmetic (rt{31}->rd{31..(31-shamt+1)}, rt{31..shamt}->rd{31-shamt..0})

R format with registers rs and rt (rd=0, shamt=0)

mult	rs, rt	R/00/18	Multiply
multu	rs, rt	R/00/19	Unsigned multiply
div	rs, rt	R/00/1A	Divide (with overflow) (rs / rt -> lo, rs % rt -> hi)
divu	rs, rt	R/00/1B	Divide (without overflow)
tge	rs, rt	R/00/30	Trap if greater equal
tgeu	rs, rt	R/00/31	Unsigned trap if greater equal
tlt	rs, rt	R/00/32	Trap if less than
tltu	rs, rt	R/00/33	Unsigned trap if less than
teq	rs, rt	R/00/34	Trap if equal
tne	rs, rt	R/00/36	Trap if not equal
madd	rs, rt	R/1C/00	Multiply add
maddu	rs, rt	R/1C /01	Unsigned multiply add
msub	rs, rt	R/1C /04	Multiply subtract
msubu	rs, rt	R/1C /05	Unsigned multiply subtract

R format with registers rd and rs (rt=rd, shamt=0)

clz	rd, rs	R/1C/20	Count leading zeros
clo	rd, rs	R/1C/21	Count leading ones

R format with registers rs and optional rd that defaults to 31 (rt=0, shamt=0)

jalr	rs, rd	R/00/09	Jump and link register

R format with rd register (rs=0, rt=0, shamt=0)

mfhi	rd	R/00/10	Move from hi ($hi->rd)
mflo	rd	R/00/12	Move from lo ($lo->rd)

R format with rs register (rd=0, rt=0, shamt=0)

jr	rs	R/00/08	Jump register
mthi	rs	R/00/11	Move to hi
mtlo	rs	R/00/13	Move to lo

R format with rd register holding an integer value (rs=0, rt=0, shamt=0)

break	code	R/00/0D	Break

R format with no registers (rs=0, rd=0, rt=0, shamt=0)

nop		R/00/00	No operation
syscall		R/00/0C	System call

I format with registers rt and rs, and immediate value

addi	rt, rs, imm	I/08	Addition immediate (with overflow) signed (rs + imm -> rt)
addiu	rt, rs, imm	I/09	Addition immediate (without overflow) signed
slti	rt, rs, imm	I/0A	Set less than immediate
sltiu	rt, rs, imm	I/0B	Set less than unsigned immediate
andi	rt, rs, imm	I/0C	AND immediate unsigned (rs{15,0} & imm -> rt{15,0}, 0 -> rt{31,16})
ori	rt, rs, imm	I/0D	OR immediate unsigned (rs{15,0} | imm -> rt{15,0}, 0 -> rt{31,16})
xori	rt, rs, imm	I/0E	Exclusive OR immediate unsigned (rs{15,0} xor imm -> rt{15,0}, 0 -> rt{31,16})

I format with registers rs and rt, and branch label

beq	rs, rt, label	I/04	Branch on equal (if rs == rt then pc + label -> pc)
bne	rs, rt, label	I/05	Branch on not equal (if rs != 0 then pc + label -> pc)

I format with rs register and branch label and register rt holding a kind of function code

bltz	rs, label	I/01,rt=00	Branch on less than zero (if rs < 0 then pc + label -> pc)
bgez	rs, label	I/01,rt=01	Branch on greater than equal zero (if rs >= 0 then pc + label -> pc)
bgezal	rs, label	I/01,rt=11	Branch on greater than equal zero and link
blez	rs, label	I/06,rt=00	Branch on less than equal zero (if rs <= 0 then pc + label -> pc)
bgtz	rs, label	I/07,rt=00	Branch on greater than zero (if rs > 0 then pc + label -> pc)
bltzal	rs, label	I/07,rt=10	Branch on less than and link

I format with rt register and immediate value (rs=0)

lui	rt, imm	I/0F	Load upper immediate unsigned (imm->rt{31..16}, 0->rt{15..0})

I format with rs register, immediate value, and register rt holding stated value

tgei	rs, imm	I/01,rt=08	Trap if greater equal immediate
tgeiu	rs, imm	I/01,rt=09	Unsigned trap if greater equal immediate
tlti	rs, imm	I/01,rt=0A	Trap if less than immediate
tltiu	rs, imm	I/01,rt=0B	Unsigned trap if less than immediate
teqi	rs, imm	I/01,rt=0C	Trap if equal immediate
tneqi	rs, imm	I/01,rt=0E	Trap if not equal immediate

I format load/store with rt register and immediate value and rs register representing the address a

lb	rt, address(rs)	I/20	Load byte (mem[rs + addr]->rt{7..0}; rt{7}->rt{31..8})
lh	rt, address(rs)	I/21	Load half-word (mem[rs + addr]->rt{15..0}; rt{15}->rt{31..16})
lw	rt, address(rs)	I/23	Load word (mem[rs + addr] -> rt)
lbu	rt, address(rs)	I/24	Load unsigned byte (0->rt{31..8}, mem[rs + address]->rt{7..0})
lhu	rt, address(rs)	I/25	Load unsigned halfword (mem[rs + addr]->rt{31..16}, 0->rt{15..0})
sb	rt, address(rs)	I/28	Store byte (rt{7..0}->mem[rs + addr])
sh	rt, address(rs)	I/29	Store halfword ({15..0}->mem[rs + addr])
sw	rt, address(rs)	I/2B	Store word (rt->mem[rs + addr])

J format

j	target	J/02	Jump (target->pc)
jal	target	J/03	Jump and link (pc->ra, target->pc)

Pseudocode

neg	rdest, rsrc	sub rdest, $zero, rsrc	Negate value (with overflow)
negu	rdest, rsrc	subu rdest, $zero, rsrc	Negate value (without overflow)
move	rdest, rsrc	addu rdest, $zero, rsrc	Move
not	rdest, rsrc	nor rdest, rsrc, $zero	NOT

abs	rdest, rsrc	addu rdest, $zero, rsrc
		bgez rdest 8
		sub rdest, $zero, rsrc
			Absolute value

la	rdest, address	lui $at, address{31..16} ori rdest, $at, address{15..0}	Load address

li	rdest, imm	if (imm >= 0)
		ori rdest, $zero, imm
		else
		lui $at, -1
		ori rdest, $at, imm
			Load immediate

beqz	rsrc, label	beq rsrc, $zero, label	Branch on equal zero
bnez	rsrc, label	bne rsrc, $zero, label	Branch on not equal zero
bge	rsrc1, rsrc2, label	slt $at, rsrc1, rsrc2 beq $at, $zero, label	Branch on greater than equal
bgeu	rsrc1, rsrc2, label	sltu $at, rsrc1, rsrc2 beq $at, $zero, label	Branch on greater than equal unsigned
ble	rsrc1, rsrc2, label	slt $at, rsrc2, rsrc1 beq $at, $zero, label	Branch on less than equal
bleu	rsrc1, rsrc2, label	sltu $at, rsrc2, rsrc1 beq $at, $zero, label	Branch on less than equal unsigned
bgt	rsrc1, rsrc2, label	slt $at, rsrc2, rsrc1 bne $at, $zero, label	Branch on greater than
bgtu	rsrc1, rsrc2, label	sltu $at, rsrc2, rsrc1 bne $at, $zero, label	Branch on greater than unsigned
blt	rsrc1, rsrc2, label	slt $at, rsrc1, rsrc2 bne $at, $zero, label	Branch on less than
bltu	rsrc1, rsrc2, label	sltu $at, rsrc1, rsrc2 bne $at, $zero, label	Branch on less than unsigned
mulo rdest, rsrc1, src2		ori $at, $zero, src2 mult rsrc1, $1 mfhi $at mflo rdest sra rdest, rdest, 31 beq $at, rdest, 8 break $0 mflo rdest	Multiply (with overflow)
mulou rdest, rsrc1, src2		ori $at, $zero, src2 multu rsrc1, $1 mfhi $at beq $at, $zero, 8 break $0 mflo rdest	Unsigned multiply (with overflow)

rem rdest, rsrc1, rsrc2	bne rsrc2, $zero, 8 break $zero div rsrc1, rsrc2 mfhi rdest	Remainder
remu rdest, rsrc1, rsrc2	bne rsrc2, $zero, 8 break $zero divu rsrc1, rsrc2 mfhi rdest	Unsigned remainder
rol rdest, rsrc1, rsrc2	subu $at, $zero, rsrc2 srlv $at, rsrc1, $at sllv rdest, rsrc1, rsrc2 or rdest, rdest, $at	Rotate left
ror rdest, rsrc1, rsrc2	subu $at, $zero, rsrc2 sllv $at, rsrc1, $at srlv rdest rsrc1, rsrc2 or rdest, rdest, $at	Rotate right
seq rdest, rsrc1, rsrc2	beq rsrc2, rsrc1, 12 ori rdest, $zero, 0 beq $zero, $zero, 8 ori rdest, $zero, 1	Set equal
sge rdest, rsrc1, rsrc2	bne rsrc2, rsrc1, 12 ori rdest, $zero, 1 beq $zero, $zero, 8 slt rdest, rsrc2, rsrc1	Set greater than equal
sgeu rdest, rsrc1, rsrc2	bne rsrc2, rsrc1, 12 ori rdest, $zero, 1 beq $zero, $zero, 8 sltu rdest, rsrc2, rsrc1	Set greater than equal unsigned
sgt rdest, rsrc1, rsrc2	slt rdest, rsrc2, rsrc1	Set greater than
sgtu rdest, rsrc1, rsrc2	sltu rdest, rsrc2, rsrc1	Set greater than unsigned
sle rdest, rsrc1, rsrc2	bne rsrc2, rsrc1, 12 ori rdest, $zero, 1 beq $zero, $zero, 8 slt rdest, rsrc1, rsrc2	Set less than equal
sleu rdest, rsrc1, rsrc2	bne rsrc2, rsrc1, 12 ori rdest, $zero, 1 beq $zero, $zero, 8 sltu rdest, rsrc1, rsrc2	Set less than equal unsigned
sne rdest, rsrc1, rsrc2	beq rsrc2, rsrc1, 12 ori rdest, $zero, 1 beq $zero, $zero, 8 ori rdest, $zero, 0	Set not equal

Pseudocode Extending Assembly Instructions

div rdest, rsrc1, src2	ori $at, $zero, src2 div rsrc1, $at mflo rdest	Divide (with overflow)
divu rdest, rsrc1, src2	ori $at, $zero, src2 divu rsrc1, $at mflo rdest	Divide (without overflow)
mul rdest, rsrc1, src2	ori $at, $zero, src2 mul rdest, rsrc1, $at	Multiply (without overflow)