MIPS reference sheet for FIT1008 and FIT2085

Semester 1, 2019

Table 1: System calls

Call code (`$v0`)	Service	Arguments	Returns	Notes
1	Print integer	`$a0` = value to print	-	value is signed
4	Print string	`$a0` = address of string to print	-	string must be terminated with `'\0'`
5	Input integer	-	`$v0` = entered integer	value is signed
8	Input string	`$a0` = address at which the string will be stored `$a1` = maximum number of characters in the string	–	returns if `$a1-1` characters or Enter typed, the string is terminated with `'\0'`
9	Allocate memory	`$a0` = number of bytes	`$v0` = address of first byte	-
10	Exit	-	-	ends simulation

Table 2: General-purpose registers

Number	Name	Purpose
R00	`$zero`	provides constant zero
R01	`$at`	reserved for assembler
R02, R03	`$v0, $v1`	system call code, return value
R04–R07	`$a0--$a3`	system call and function arguments
R08–R15	`$t0--$t7`	temporary storage (caller-saved)
R16–R23	`$s0--$s7`	temporary storage (callee-saved)
R24, R25	`$t8, $t9`	temporary storage (caller-saved)
R28	`$gp`	pointer to global area
R29	`$sp`	stack pointer
R30	`$fp`	frame pointer
R31	`$ra`	return address

Table 3: Assembler directives

.data	assemble into data segment
.text	assemble into text (code) segment
.word w1[, w2, ...]	allocate word(s) with initial value(s)
.space n	allocate n bytes of uninitialized, unaligned space
.ascii "string"	allocate ASCII string, do not terminate
.asciiz "string"	allocate ASCII string, terminate with `'\0'`

Table 4: Function calling convention

	Caller:	**Callee:**
On function call:	saves temporary registers on stack passes arguments on stack calls function using `jal fn_label`	saves value of `$ra` on stack saves value of `$fp` on stack copies `$sp` to `$fp` allocates local variables on stack

	Callee:	**Caller:**
On function return:	sets `$v0` to return value clears local variables off stack restores saved `$fp` off stack restores saved `$ra` off stack returns to caller with `jr $ra`	clears arguments off stack restores temporary registers off stack uses return value in `$v0`

Table 5: Allowed MIPS instruction (and pseudoinstruction) set

Instruction format	Meaning	Operation	Immediate	Unsigned format
add Rdest, Rsrc1, Src2	Add	Rdest = Rsrc1 + Src2	addi	addu (no overflow trap)
sub Rdest, Rsrc1, Src2	Subtract	Rdest = Rsrc1 - Src2	-	subu (no overflow trap)
mult Rsrc1, Src2	Multiply	Hi:Lo = Rsrc1 * Src2	-	mulu
div Rsrc1, Src2	Divide	Lo = Rsrc1/Src2; Hi = Rsrc1 % Src2	-	divu
and Rdest, Rsrc1, Src2	Bitwise AND	Rdest = Rsrc1 & Src2	andi	-
or Rdest, Rsrc1, Src2	Bitwise OR	Rdest = Rsrc1 | Src2	ori	-
xor Rdest, Rsrc1, Src2	Bitwise XOR	Rdest = Rsrc1 $\wedge$ Src2	xori	-
nor Rdest, Rsrc1, Src2	Bitwise NOR	Rdest = $\sim$(Rsrc1 | Src2)	-	-
sllv Rdest, Rsrc1, Src2	Shift Left Logical	Rdest = Rsrc1 << Src2	sll	-
srlv Rdest, Rsrc1, Src2	Shift Right Logical	Rdest = Rsrc1 >> Src2 (MSB=0)	srl	-
srav Rdest, Rsrc1, Src2	Shift Right Arithmetic	Rdest = Rsrc1 >> Src2 (MSB preserved)	sra	-
mfhi Rdest	Move from Hi	Rdest = Hi	-	-
mflo Rdest	Move from Lo	Rdest = Lo	-	-
lw Rdest, Addr	Load word	Rdest = mem32[Addr]	-	-
sw Rsrc, Addr	Store word	mem32[Addr] = Rsrc	-	-
la Rdest, Addr(or label)	Load Address (for printing strings)	Rdest=Addr (or Rdest=label)	-	-
beq Rsrc1, Rsrc2, label	Branch if equal	if (Rsrc1 == Rsrc2) PC = label	-	-
bne Rsrc1, Rsrc2, label	Branch if not equal	if (Rsrc1 != Rsrc2) PC = label	-	-
slt Rdest, Rsrc1, Src2	Set if less than	if (Rsrc1 < Src2) Rdest = 1 else Rdest = 0	slti	sltu
j label	Jump	PC = label	-	-
jal label	Jump and link	$ra = PC + 4; PC = label	-	-
jr Rsrc	Jump register	PC = Rsrc	-	-
jalr Rsrc	Jump and link register	$ra = PC + 4; PC = Rsrc	-	-