RISC-V/RARS General Purpose Registers

0	zero	all bits are zero	16	a6	argument
1	ra	return address	17	a7	argument
2	sp	stack pointer	18	s2	local variable
3	gp	global pointer	19	s3	local variable
4	tp	thread pointer	20	s4	local variable
5	t0	temporary	21	s5	local variable
6	t1	temporary	22	s6	local variable
7	t2	temporary	23	s7	local variable
8	s0	local variable	24	s8	local variable
9	s1	local variable	25	s9	local variable
10	a0	argument / return value	26	s10	local variable
11	a1	argument / return value	27	s11	local variable
12	a2	argument	28	t3	temporary
13	a3	argument	29	t4	temporary
14	a4	argument	30	t5	temporary
15	a5	argument	31	t6	temporary

Remarks: (1) Only `a0` is used for a return value in cases where the return value fits in a single 32-bit GPR. (2) In some cases it makes sense to use t-registers instead of s-registers for local variables. (3) In some cases it makes sense to use s-registers for items that are not local variables of functions.

Midterm #1 Instruction Subset

The notation (*pi*) indicates a pseudoinstruction. It is fine to use pseudoinstructions on midterms and the final exam.

Addition and subtraction

- `add` *rdest, rsrc1, rsrc2* —add register *rsrc1* and register *rsrc2*; put result in register *rdest*.
- `addi` *rdest, rsrc1, imm* —add register *rsrc1* and constant *imm*; put result in register *rdest*. Constant must be in the range $-2^{11} \leq imm \leq 2^{11} - 1$, that is, $-2,048 \leq imm \leq 2,047$.
- `sub` *rdest, rsrc1, rsrc2* —subtract register *rsrc2* from register *rsrc1*; put result in register *rdest*.

Data copying

- `lw` *rdest, address* —copy word from memory at *address* to register *rdest*.
- `sw` *rsrc, address* —copy word from register *rsrc* to memory at *address*.
- `lb` *rdest, address* —copy byte from memory at *address* to bits 7–0 of register *rdest*, make bits 31–8 of *rdest* equal to bit 7 of that byte.
- `lbu` *rdest, address* —copy byte from memory at *address* to bits 7–0 of register *rdest*, make bits 31–8 of *rdest* equal to 0.
- `sb` *rsrc, address* —copy byte from bits 7–0 of register *rsrc* to memory at *address*.
- `mv` *rdest, rsrc* —(*pi*) copy contents of register *rsrc* into register *rdest*.
- `li` *rdest, imm* —(*pi*) copy constant *imm* into register *rdest*.
- `la` *rdest, label* —(*pi*) copy address corresponding to *label* into register *rdest*.

Branches and jumps

`blt`, `bge`, `ble` and `bgt` all compare register contents as signed integers. RISC-V has instructions to branch based on comparison of unsigned integers, but those are not needed for Midterm #1.

- `beq` *rsrc1, rsrc2, label* —branch to instruction at *label* if value of register *rsrc1* equals value of register *rsrc2*.
- `bne` *rsrc1, rsrc2, label* —branch to instruction at *label* if value of register *rsrc1* does not equal value of register *rsrc2*.
- `blt` *rsrc1, rsrc2, label* —branch to instruction at *label* if value of register *rsrc1* is strictly less than value of register *rsrc2*.
- `bge` *rsrc1, rsrc2, label* —branch to instruction at *label* if value of register *rsrc1* is greater than or equal to value of register *rsrc2*.
- `ble` *rsrc1, rsrc2, label* —(*pi*) branch to instruction at *label* if value of register *rsrc1* is less than or equal to value of register *rsrc2*.
- `bgt` *rsrc1, rsrc2, label* —(*pi*) branch to instruction at *label* if value of register *rsrc1* is strictly greater than value of register *rsrc2*.
- `j` *label* —(*pi*) jump to instruction at *label*.
- `jal` *label* —(*pi*) copy the address of the instruction following the `jal` instruction to `ra`, and jump to the instruction at *label*.
- `jr` *reg* —(*pi*) jump to the instruction at the address in register *reg*.

Logical instructions

- `slli` *rdest, rsrc1, count* —shift value of register *rsrc1* left by *count* bits, filling with 0's from the right; put result in register *rdest*.
- `srli` *rdest, rsrc1, count* —shift value of register *rsrc1* right by *count* bits, filling with 0's from the left; put result in register *rdest*.
- `and` *rdest, rsrc1, rsrc2* —do a bitwise AND of register *rsrc1* and register *rsrc2*, put result in register *rdest*.
- `andi` *rdest, rsrc1, imm* —do a bitwise AND of register *rsrc1* and constant *imm*, put result in register *rdest*. Constant must be in the range $-2^{11} \leq imm \leq 2^{11} - 1$, that is, $-2,048 \leq imm \leq 2,047$.
- `or` *rdest, rsrc1, rsrc2* —do a bitwise OR of register *rsrc1* and register *rsrc2*, put result in register *rdest*.
- `ori` *rdest, rsrc1, imm* —do a bitwise OR of register *rsrc1* and constant *imm*, put result in register *rdest*. Constant must be in the range $-2^{11} \leq imm \leq 2^{11} - 1$, that is, $-2,048 \leq imm \leq 2,047$.
- `xor` *rdest, rsrc1, rsrc2* —do a bitwise XOR of register *rsrc1* and register *rsrc2*, put result in register *rdest*.
- `xori` *rdest, rsrc1, imm* —do a bitwise XOR of register *rsrc1* and constant *imm*, put result in register *rdest*. Constant must be in the range $-2^{11} \leq imm \leq 2^{11} - 1$, that is, $-2,048 \leq imm \leq 2,047$.
- `not` *rdest, rsrc* —(*pi*) do a bitwise NOT of register *rsrc*, put result in register *rdest*.
- `lui` *rdest, imm* —copy constant *imm* to bits 31–12 of register *rdest*, make bits 11–0 of *rdest* 0. Constant must be in the range $0 \leq imm \leq 2^{20} - 1$, that is, $0 \leq imm \leq$ `0xfffff`.