Tutorial 11
CSE 112 Computer Organisation

Q1. What is a caller-callee convention, and why is it needed?
Ans: Caller-Callee conventions are a set of rules followed by the compiler while implementing functions. These conventions outline the compiler behavior regarding where/how arguments are placed, where/how the return address is stored, which all registers should be preserved by the caller and callee, etc.
If all of the source code is available to the compiler statically, then the compiler can come up with its own custom conventions. In this case, the caller-callee conventions are not strictly necessary.
However, often, a lot of pre-compiled libraries are linked with the source code (both statically and dynamically). Therefore there is a need to strictly define the interface these libraries provide. This interface is encapsulated in the caller-callee conventions.
The most important rules that a caller-callee convention outlines are:
1. Where are the input arguments stored by the caller before callee is called?
2. What order are the arguments stored in?
3. Where is the return address stored by the caller before calling the callee?
4. How does the callee return data to the caller?
5. Which all registers must be preserved by the caller across a function call? 6. What all registers must be preserved by the callee across a function call? There needs to be a consensus regarding who preserves what registers, as each function assumes access to all registers. If there is no consensus, then it can happen that the callee thrashes a register that the caller was using without the knowledge of the caller.

Q2. Assuming that each function is equally likely to use every register, what should be the ideal split between the number of callee-saved and caller-saved registers? Ans: Approximately 50-50 split. Since each function is equally likely to use each register, giving preference to either callee or the caller is not correct.
Suppose that all registers are caller saved. In this case, we will end up saving a lot of registers conservatively on each call, even if the callee is not using those registers. On the other hand, if only half of the registers are caller saved, the potential to "over-save" is less. Similarly, "under-saving" forces the callee to act conservatively as it limits the number of available registers which the callee can use, leaving the function called very few options in terms of number of registers to exploit for its operation.
Therefore when no other information is available, a 50-50 split between caller and callee saved registers is the most performant.

Q3. What is a stack?
Ans: The stack is a memory region that saves all the activation blocks in a program.
- It is traditionally considered to be downward growing.
- Before calling a function, we need to push its activation block to the stack ● When a function finishes execution, we need to pop its activation block off the stack.

ISA Description and Caller-Callee conventions for Q4:

Name	Semantics	Syntax
Add	Performs rd = rs1 + rs2	add rd rs1 rs2
Sub	Performs rd = rs1 - rs2	sub rd rs1 rs2
Add Immediate	Performs rd = rs + Imm	addi rd, rs, #Imm
Branch if not equal	If content of rs1 is not equal to content of rs2 then branch to label	bne rs1 rs2 **Label**

Jump and link	Jumps to label after saving the return address to rd.
	rd=PC+4
	PC=PC+sext(imm[20:1], 1 'b0}
		jal rd, **Label**

Jump and Link Register	Jump to the address formed by adding the source register and immediate after saving the return address to rd.
	rd=PC+4
	PC = rs1 + sext(imm[11:0])
		Jalr rd,rs1,imm[11:0]

7.2.8 Register Encoding

Address	Register	ABI Name	Description	Saver
5'b0000_0	x0	zero	Hard-wired zero	– – – ––
5'b0000_1	x1	ra	Return address	Caller
5'b0001_0	x2	sp	Stack Pointer	Callee
5'b0001_1	x3	gp	Global Pointer	– – – ––
5'b0010_0	x4	tp	Thread Pointer	– – – ––
5'b0010_1	x5	t0	Temporary/alternate link register	Caller
5'b00_{110,111}	x6-7	t1-2	Temporaries	Caller
5'b0100_0	x8	s0/fp	Saved register/frame pointer	Callee
5'b0100_1	x9	s1	Saved Register	Callee
5'b0101_{0,1}	x10-11	a0-1	Function arguments/ return values	Caller
(5'b011_{00-11}),(5'b1000_{0,1})	x12-17	a2-7	Function arguments	Caller
5'b1_{0010-1011}	x18-27	s2-11	Saved registers	Caller
5'b111_{00-11}	x28-31	t3-6	Temporaries	Caller

- **There are 32 general purpose registers x0-x31.**
- **ra is the return address register.**
- **a0,a1 stores the return value.**
- **The first two arguments to the callee are stored in registers a2 and a3. ● All the registers from s2-s11 are caller-saved. On the other hand, registers s0-s1 are callee saved.**
- **Whenever the branch and link instruction is used, the return address is stored in ra, and the program counter jumps to the given label.**

Q4.Given the C code, convert the following into assembly language

```
void bar()
{
        c = baz(1, 2);
        d = c+3
}
int baz(int a, int b)
{
        return a + b;
}
```

Ans:

```
baz:
        add t0 a2 a3            // using temporary register to perform computation
                        // using the values passed in register to perform computation
        Addi a0,t0,#0           // storing the value in return value register
        Jalr x0, ra, #0         // returning to the address stored in ra.
bar:
        movi a2 $1
        movi a3 $2
        jal ra, baz
        Addi s1, a0, #3
```