Chapter 2 Processor Architecture…Solutions

1. Having a large register-file is detrimental to the performance of a processor since it results in a large overhead for procedure call/return in high-level languages. Do you agree or disagree? Give supporting arguments.

   Disagree.
   By judicious use of calling conventions defining saved and temporary registers call/return overhead can be managed to any desired level of performance

2. Distinguish between the frame pointer and the stack pointer.

   The stack pointer points to the top of the stack and as more or less stack space is needed will change position during the execution of a particular piece of code (e.g. function) . Since it is common to reference the location of variables stored on the stack as an offset from the stack pointer this movement becomes problematic. A solution is to use another register to store the location of a fixed reference point that will be constant throughout the execution of some particular piece of code (e.g. function). Note: This is not to say that when a function calls another function the frame pointer will remain fixed. It will not. Rather it will be changed on call and reestablished upon return thus for all execution of an given functions own code it will be fixed.

3. In the LC-2200 architecture, where are operands normally found for an add instruction?

   The operands are in registers.

4. Endianness: Let's say you want to write a program for comparing two strings. You have a choice of using a 32-bit byte-addressable Big-endian or Little-endian architecture to do this. In either case, you can pack 4 characters in each word of 32-bits. Which one would you choose and how will you write such a program? [Hint: Normally, you would do string comparison one character at a time. If you can do it a word at a time instead of a character at a time, that implementation will be faster.]

   The choice of endianness should not matter, so long as you are consistent in the comparison. Subtract each word in string A from string B. If you return zero, the strings are identical. Note, you can only perform this operation on the same system. The operation will be more complex if you are trying to compare a Big-endian system to a Little-endian system. You would have to compare character by character.

5. ISA may support different flavors of conditional branch instructions such as BZ (branch on Zero), BN (branch on negative), and BEQ (branch on equal). Figure out the predicate expressions in an "if" statement that may be best served by these different flavors of conditional branch instructions. Give examples of such predicates in "if" statement, and how you will compile them using these different flavors of branch instructions.

   BZ: if (a == 0)
   BEQ: if (a == b)
   BN: if (a < 0) or if (a < b) which becomes if (a-b < 0)
   BP: if (a > 0) or if (a>b) which becomes if(a-b > 0)

6. We said that endianness will not affect your program performance or correctness so long as the use of a (high level) data structure is commensurate with its declaration. Are there situations where even if your program does not violate the above rule, you could be bitten by the endianness of the architecture? [Hint: Think of programs that cross network boundaries.]

   Yes, if data from a big-endian computer were transferred over a network to a small endian computer data corruption could occur. However, this problem has largely been solved with the Internet and networks which use similar technology. The solution is for networks to use a standard endianness. If the endianness of the network is different from the host computer, the host's network interface will apply the appropriate conversion.

7. Work out the details of the implementing the switch statement using jump tables in assembly using any flavor of conditional branch instruction. [Hint: After ensuring that the value of the switch variable is within the bounds of valid case values, jump to the start of the appropriate code segment corresponding to the current switch value, execute the code and finally jump to exit.]

   - First, check the bounds of the switch variable
   - If the variable is within the bounds, you would index into the jump-table based on which switch case was executed. For example, if you took the second switch statement, you would jump to the second location listed in the jump-table.
   - Execute the desired branch
   - JLR or JMP $returnaddress
   - Continue with your original code

8. Procedure A has important data in both S and T registers and is about to call procedure B. Which registers should A store on the stack? Which registers should B store on the stack?

   Procedure A should save the T registers before calling procedure B. B should save any S registers it uses OR save any T registers it needs before calling another function.

9. Consider the usage of the stack abstraction in executing procedure calls. Do all actions on the stack happen only via pushes and pops on to and from the top of the stack? Explain circumstances that warrant reaching into other parts of the stack during program execution. How is this accomplished?

   No, the amount of memory included in the stack at any given time is controlled by simply changing the stack pointer value. Then values may be read or written in locations defined as offsets from the address stored in the stack pointer (or frame pointer).

10. Answer True/False with justification: Procedure call/return cannot be implemented without a frame pointer.

    False, the frame pointer is not necessary to implement procedure calls, but it can make code simpler.

11. DEC VAX had a single instruction for loading and storing all the program visible registers from/to memory. Can you see a reason for such an instruction pair? Consider both the pros and cons.

    Pros: If you are a callee and need to use all of the temporary and safe registers, you can perform this operation in one call. If you want to save the current state of execution, it can be done in one call.

    Cons: In most cases you do not need all available registers, so this command uses more memory (and possibly time) than required.

12. Show how you can simulate a subtract instruction using the existing LC-2200 ISA?

    Since our system uses 2's complement, the negative value of a number is NOT X plus 1. The LC-2200 does not have support for NOT, but NAND serves the same function.

```
B ← B NAND B    ; not B
B ← B+1         ; B+1
A ←A+B          ; A+B, net result is A-B
```

13. The BEQ instruction restricts the distance you can branch to from the current position of the PC. If your program warrants jumping to a distance larger than that allowed by the offset field of the BEQ instruction, show how you can accomplish such “long” branches using the existing LC-2200 ISA.

```
        BEQ     $s0, $s1, Near
        BEQ     $zero, $zero, Skip
Near    JALR    $s2, $zero
Skip    …
```

Note: Assume the address of the location that is a "long" way away is in $s2

14. What is an ISA and why is it important?

The ISA (Instruction Set Architecture) serves as a kind of contractual document which allows all parties concerned with the design, implementation and use of a given processor to know what is expected of them and what resources will be afforded by that processor.

As soon as an ISA is finalized:

- Implementation engineers can design the detail that will allow the processor to meet the ISA specification
- Assembler and compiler writers can create assemblers and compiler for use with the processor long before a working model even exists.
- Operating system designers/maintainers can determine what needs to be done to allow their operating system to run on this processor.
- I/O Device engineers can design controllers and driver software that will be used with the processor.
- Box (or equivalent) engineers can determine how to use the processor in their designs
- etc.

15. What are the influences on instruction set design?

Instruction sets are influenced by efficiency, ease of implementation, and ease of programming. A larger instruction set makes writing compilers easier, but at the detriment of speed or ease of implementation. See also CISC and RISC.

16. What are conditional statements and how are they handled in the ISA?

Conditional statements compare 2 values to determine some sort of relation (equality, equal to zero, positive, or negative). The ISA specifies which conditional statements are available. The LC-2200 features BEQ, but the LC-2110 had BR(N/Z/P).

17. Define the term addressing mode.

    An addressing mode specifies how the bits of the instruction determine the location of the operands. For example, some of the bits might be a register number or an offset to be added to the PC, etc.

18. In Section 2.8, we mentioned that local variables in a procedure are allocated on the stack. While this description is convenient for keeping the exposition simple, modern compilers work quite differently. This exercise is for you to search the Internet and find out how exactly modern compilers allocate space for local variables in a procedure call. [Hint: Recall that registers are faster than memory. So, the objective should be to keep as many of the variables in registers as possible.]

    As we have already seen many local variables which technically if located in memory would be found on the stack are in reality maintained in registers because of the significant advantage in speed enjoyed by the registers. We have already seen saved and temporary register conventions, argument registers, return value and return address registers. All of these are an attempt to increase speed and efficiency. In addition, modern optimizing compilers employ sophisticated register allocation strategies designed to maximize use of registers. However, arrays and structures are maintained on the stack and not in registers.

19. We use the term abstraction to refer to the stack. What is meant by this term? Does the term abstraction imply how it is implemented? For example, is a stack used in a procedure call/return a hardware device or a software device?

    An abstraction is a way of simplifying and eliminating unnecessary details while defining the desired behavior of a system. Normally abstraction implies hiding implementation details. For example, a queue is an abstraction that has to support enqueuing and dequeuing. The queue abstraction may be implemented with a linked list, an array, etc. The stack facilitating a procedure call/return is a software abstraction implemented with memory and register hardware but it could also be implemented as a separate device.

20. Given the following instructions

```
BEQ Rx, Ry, offset    ; if (Rx == Ry) PC=PC+offset
SUB Rx, Ry, Rz        ; Rx <- Ry - Rz
ADDI Rx, Ry, Imm      ; Rx <- Ry + Immediate value
AND Rx, Ry, Rz        ; Rx <- Ry AND Rz
```

    Show how you can realize the effect of the following instruction:

```
BGT Rx, Ry, offset    ; if (Rx > Ry) PC=PC+offset
```

Assume that the registers and the Imm field are 8-bits wide. You can ignore the case that the SUB instruction causes an overflow.

Solution:

If Rx > Ry then Ry-Rx < 0

```
SUB   $at, Ry, Rx               ; Ry - Rx
ADDI  $t3, $zero, x80           ; Create the mask 1000 0000
AND   $at, $at, $t3             ; Check for negative
BEQ   $at, $zero offset

Note: $t3 would need to be saved if in use already
```

21. Given the following load instruction

```
LW Rx, Ry, OFFSET     ; Rx <- MEM[Ry + OFFSET]
```

Show how to realize a new addressing mode, called *indirect*, for use with the load instruction that is represented in assembly language as:

```
LW Rx, @(Ry) ;
```

The semantics of this instruction is that the contents of register Ry is the address of a pointer to the memory operand that must be loaded in Rx.

Solution:

```
LW Rx, Ry, 0
LW Rx, Rx, 0
```

22. Convert this statement:

```
g = h + A[i];
```

into an LC-2200 assembler with the assumption that the address of A is located in $t0, g is in $s1, h is in $s2, and, i is in $t1.

```
ADD   $t2, $t0, $t1        ; Calculate address of A[i]
LW    $t3, 0($t2)          ; Load the contents of A[i] into a register
ADD   $s1, $s2, $t3        ; Assign sum to g
```

23. Suppose you design a computer called the Big Looper 2000 that will never be used to call procedures and that will automatically jump back to the beginning of memory when it reaches the end. Do you need a program counter? Justify your answer.

Even the Big Looper 2000 will need a PC in order to know from what address to fetch an instruction from on each cycle. The PC will also be used in the calculation of PC relative addressing such as those used with branch instructions.

24. Consider the following program and assume that for this processor:

- All arguments are passed on the stack.
- Register V0 is for return values.
- The S registers are expected to be saved, that is a calling routine can leave values in the S registers and expect it to be there after a call.
- The T registers are expected to be temporary, that is a calling routine must not expect values in the T registers to be preserved after a call.

```
int bar(int a, int b)
{
    /* Code that uses registers T5, T6, S11-S13; */
    return(1);
}

int foo(int a, int b, int c, int d, int e)
{
    int x, y;
    /* Code that uses registers T5-T10, S11-S13; */
    bar(x, y); /* call bar */
    /* Code that reuses register T6 & arguments a, b, & c; */
    return(0);
}

main(int argc, char **argv)
{
    int p, q, r, s, t, u;
    /* Code that uses registers T5-T10, S11-S15; */
    foo(p, q, r, s, t); /* Call foo */
    /* Code that reuses registers T9, T10; */
}
```

Here is the stack when bar is executing, clearly indicate in the spaces provided which procedure (main, foo, bar) saved specific entries on the stack.

main	foo	bar	
_X__	____	____	p
_X__	____	____	q

_X__	_____	_____	r
_X__	_____	_____	s
_X__	_____	_____	t
_X__	_____	_____	u
_X__	_____	_____	T9
_X__	_____	_____	T10
_____	_X__	_____	p
_____	_X__	_____	q
_____	_X__	_____	r
_____	_X__	_____	s
_____	_X__	_____	t
_____	_X__	_____	x
_____	_X__	_____	y
_____	_X__	_____	S11
_____	_X__	_____	S12
_____	_X__	_____	S13
_____	_____	_____	S14
_____	_____	_____	S15
_____	_X__	_____	T6
_____	_____	_X__	x
_____	_____	_X__	y
_____	_____	_X__	S11
_____	_____	_X__	S12
_____	_____	_X__	S13 <----------- Top of Stack