Matching. Indicate the letter of the best description. (1 pt. each)

1. __b__ address
   a. operand for addition
   b. name of memory location
   c. human-readable, symbolic program
   d. binary code that will need linking
   e. binary code ready for execution
   f. specifies interrupt condition
   g. reflects summary outcome of last recorded ALU operation
   h. 4 bits
   i. 8 bits
   j. floating-point number encoding
   k. character set encoding
   l. variable dynamically allocated in heap
   m. variable allocated in stack frame
   n. variable allocated with in .data/.bss
   o. allocates space for global variable
   p. declares external symbol (use or def)
   q. resumes use of global location counter
   r. controller registers have main memory addresses
   s. loads and stores can access controller registers
   t. special IN and OUT instructions access controller registers
   u. a speed-matching buffer for bursty I/O

2. __d__ object code
   v. copy address of actual parameter into space allocated for formal parameter

3. __g__ condition code
   w. copy value of actual parameter into space allocated for formal parameter

4. __i__ byte
   x. copy final value of formal parameter back into actual parameter

5. __k__ ASCII
   y. copy value of actual parameter into space allocated for formal parameter, and copy final value of formal parameter back into actual parameter

6. __m__ local variable
   z. copy address of formal parameter into space allocated for actual parameter

7. __p__ .global

8. __t__ isolated I/O (port I/O)

9. __w__ call by value

10. __x__ call by result

11. __y__ call by value-result

12. __y__ call by reference
13. Why are many program translators structured as two passes over the input? (2 pts.)

<ANSWER>

forward references (use of symbols prior to their definition) leads to a two-pass (or one-pass-with-fixup) structure

a first pass collects all definitions into a symbol table, and
a second pass uses the symbol table in translating

14. Name a one-pass translator we have studied. (1 pt.)

<ANSWER>

a macro processor, like m4, is typically structured as a one-pass translator, requiring macros to be defined before they are used

15. What is the most negative 12-bit two's complement number? Please give both the hexadecimal representation and the decimal representation. (2 pts.)

<ANSWER>

0x800 = -2048

16. What is the most positive 12-bit two's complement number? Please give both the hexadecimal representation and the decimal representation. (2 pts.)

<ANSWER>

0x7ff = +2047

17. Convert these numbers between signed decimal and 16-bit two's complement representation. (2 pts. each)

<table>
<thead>
<tr>
<th>signed decimal</th>
<th>two's complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. -10</td>
<td>0xffff6</td>
</tr>
<tr>
<td>b. -240</td>
<td>0xff10</td>
</tr>
<tr>
<td>c. 1010</td>
<td>0x03f2</td>
</tr>
<tr>
<td>d. 4112</td>
<td>0x1010</td>
</tr>
</tbody>
</table>
18. Consider the simple 5-bit floating-point format we have used in class: one sign bit, two exponent bits with the exponent encoded in bias-2 notation, two fraction bits, and a hidden bit. Show the 5-bit values for these decimal numbers and mark them on the number line: (3 pts. each)

\[
\begin{array}{ccc}
2^{**(-1)} & 2^{*0} & 2^{*1} \\
0 & / & \backslash \\
\vdash & \vdash & \vdash \\
\vdash | \vdash | \vdash | \vdash | \vdash | \vdash | \vdash | \vdash | \vdash | \vdash |
\end{array}
\]

\begin{align*}
\text{a. } & +0.5 = \_0_ (\_01\_ ) \_00 \\
\text{b. } & +1.5 = \_0_ (\_10\_ ) \_10 \\
\text{c. } & +2.5 = \_0_ (\_11\_ ) \_01 \\
\end{align*}

Extra credit. Show the floating-point addition of 0.5 and 1.5. (up to 6 pts.)

\[
\begin{align*}
0.5 & = 1.00 \times 2^{(-1)} \rightarrow 0.10 \times 2^{(0)} \\
+1.5 & = +1.10 \times 2^{(0)} \rightarrow +1.10 \times 2^{(0)} \\
\text{-------------------------} & \text{-------------------------} \\
10.00 \times 2^{(0)} \rightarrow 1.00 \times 2^{(+1)} \\
\end{align*}
\]

2.0 is encoded as 0 (11) 00
19. Write code that implements the following loop in SPARC assembly language.
(8 pts.)

```plaintext
register int i,sum; /* can be allocated to registers */
static int a[20]; /* not on stack, assume a base address of "a" */
sum = 0;
for( i = 0; i < 20; i++ ){
    sum = sum + a[i];
}

<POSSIBLE ANSWER>

define(sum_r,l0)
define(i_r,l1)
define(base_r,l2)
define(offset_r,l3)
define(value_r,l4)

    clr %sum_r
    clr %i_r
    set a,%base_r

loop: cmp %i_r, 20
    bge done
    nop

    sll %i_r, 2, %offset_r
    ld [%base_r + %offset_r], %value_r
    add %sum_r, %value_r, %sum_r

    inc %i_r
    ba loop
    nop

done:
```
20. Show the steps required in multiplication of two unsigned 4-bit binary numbers, 1010 times 1101. (1010 is the multiplicand and 1101 is the multiplier.) The details of mulsc are not required, but you should show the 3-register format of ACC, MQ, and MDR. (6 pts. + 1 pt. extra credit for placing the multiplicand and multiplier in the correct registers.)

<ANSWER>

\[ 1010 \text{ 4-bit multiplicand} \times 1101 \text{ 4-bit multiplier} \]

 initially: C ACC MQ
 \[ 0 \quad 0000 \quad 1101 \]

 MDR
 \[ 1010 \]

 step 1: \[ 0 \quad 0000 \quad 1101 \]
 ^ add based on lsb=1

 \[ 0 \quad 1010 \quad 1101 \]
 >> shift right
 \[ 0 \quad 0101 \quad 0110 \]

 step 2: \[ 0 \quad 0101 \quad 0110 \]
 ^ no add based on lsb=0

 \[ 0 \quad 0101 \quad 0110 \]
 >> shift right
 \[ 0 \quad 0010 \quad 1011 \]

 step 3: \[ 0 \quad 0010 \quad 1011 \]
 ^ add based on lsb=1

 \[ 0 \quad 1100 \quad 1011 \]
 >> shift right
 \[ 0 \quad 0110 \quad 0101 \]

 step 4: \[ 0 \quad 0110 \quad 0101 \]
 ^ add based on lsb=1

 \[ 1 \quad 0000 \quad 0101 \]
 >> shift right
 \[ 0 \quad 1000 \quad 0010 \]

 check: \[ 1010 \quad 10 \]
 x \[ 1101 \quad x \quad 13 \]

 \[ 10000010 = 128 + 2 = 130 \]
21. Fill in the blanks for generic subroutine actions. (1 pt. per blank)

caller:
push parameter2 on stack
push parameter1 on stack
call subroutine /* pushes return address onto the stack */

subroutine: /* callee save */

push ___ registers_to_save___

push old fp

set new ___ fp ___ as current ___ sp ___

subtract from ___ sp ___ to make room for locals

<body of subroutine>

set ___ sp ___ to current ___ fp ___

pop ___ old fp ___

pop ___ registers_to_save ___

return /* pops return address from stack */

clean parameters off stack

22. Fill in the blanks for SPARC subroutine actions. (1 pt. per blank)

caller:
move parameter1 to ___%o0___

move parameter2 to ___%o1___

call subroutine /* places address of call in %o7 */

subroutine: (local variables in %l0,... regs)

___save_%sp,-96,%sp___

<body of subroutine, in which you

access parameter1 in ___%i0___ and

access parameter2 in ___%i1___ >

ret /* return to %i7+8 */

restore
23. Explain why the SPARC designers did not want to use a memory stack to pass parameters for subroutine calls and what mechanism the SPARC designers provided instead. (3 pts.)

<ANSWER>

memory access is slower than register access, so the SPARC designers provided register windows as a way to avoid load and store memory accesses during procedure calls

(note: many would argue that register windows are more complicated than they are worth and no faster than processors with cache memory, since the top of the stack is typically cache resident and fast to access)

24. (2 pts. each)
   a) What does a GHz number for a CPU represent? What is a typical value?

<ANSWER>

GHz is the CPU clock frequency, measured in $10^9$ cycles per second; 2 GHz is a typical value

b) What does a GB number for a memory represent? What is a typical value?

<ANSWER>

GB is the main memory size, measured in $2^{30}$ bytes of memory; 4 GB is a typical value

c) What does a TB number for a disk represent? What is a typical value?

<ANSWER>

TB is the disk capacity, measured in $10^{12}$ bytes of storage; 1 TB is a typical value

d) What does a seek time number for a disk represent? In what units is it measured? What is a typical value?

<ANSWER>

seek time is the time required to move the read/write arm to a particular track, measured in milliseconds ($10^{-3}$ seconds); 4 msec is a typical value

25. List the five device registers found in a DMA I/O controller. (5 pts.)

<ANSWER>

command, status, data, address, count
26. Define "logical" and "physical" records in terms of I/O transfers. (2 pts.)

<ANSWER>

a logical record is the unit of I/O access performed by an application to/from a memory buffer, and a physical record is the unit of access to/from the storage device.

logical record
  ^
  getc()/putc(), scanf()/printf(),
  cin/cout, etc.
  v
memory buffer
  ^
physical record
  ^
  read()/write(), etc.
  v
I/O device

Logical records are typically small - even down to individual character transfers - while physical records are typically large (e.g., can be a set of multiple contiguous 512-byte sectors on disk).

27. Briefly explain the concept and benefit of blocking in I/O transfers. (4 pts.)

<ANSWER>

blocking is the collection of multiple logical records per physical record; since each I/O operation has some overhead added to the actual data transfer (e.g., interrupt processing, seek and rotational latency on a disk), blocking is more efficient since the overhead is required only once per physical record rather than once for each logical record.

(compare to withdrawing cash from an ATM with a $1 per-transaction fee - it costs less overall to withdraw $20 in one transaction as opposed to two separate $10 transactions)

28. Explain the difference between raw and cooked character processing. (3 pts.)

<ANSWER>

raw mode provides every input character directly to the application, while cooked mode intercepts and interprets editing characters (such as backspace and delete)

29. Draw a diagram of the fetch/decode/execute cycle of a computer that includes the four actions that occur when interrupts are accepted and the actions for an RTI instruction. (10 pts.)
30. We discussed the pros and cons of static linking versus dynamic linking. Name two other distinct design choices we have studied (in the form of "a" versus "b") that result from a similar time versus space tradeoff. (2 pts.)

<POSSIBLE ANSWERS>

macros versus subroutine calls
(subroutines have smaller footprint in a program than in-lined macros but require runtime overhead for call-return linkage)

blocked versus unblocked I/O
(see the answer to question 27)

bucket versus cup ;)
(when getting water from a well)
Extra credit. Consider the following C program and the (slightly edited) generated assembly from gcc -S -O2.

```c
char *strcpy( char *dest, char *src ){
    char *save = dest;
    while( *dest++ = *src++ );
    return save;
}
```

```
a01:   strcpy:
a02:       mov     %o0, %g2
a03:   .LL2:
a04:       ldub    [%o1], %g1
a05:       add     %o1, 1, %o1
a06:       stb     %g1, [%g2]
a07:       add     %g2, 1, %g2
a08:       sll     %g1, 24, %g1
a09:       cmp     %g1, 0
a10:       bne     .LL2
a11:        nop
a12:       retl
a13:        nop
```

Explain why the shift in a08 is generated and whether it is actually needed or not. (up to 4 pts.)

<ANSWER>

The "char" data type is signed, so the shift is placing the sign bit of the 8-bit value in the least-significant byte of %g1 into the sign bit of the register; this is unnecessary since the branch condition is not-equal-zero rather than one which requires the N (negative) condition code to be set correctly.