Question 1:

The system in question has 1MiB of physical memory, 32-bit virtual addresses, and 256 physical pages. The memory management system uses a fully associative TLB with 128 entries and an LRU replacement scheme.

a. What is the size of the physical pages in bytes?  
   \(2^{12}\) bytes

b. What is the size of the virtual pages in bytes?  
   \(2^{12}\) bytes

c. What is the maximum number of virtual pages a process can use?  
   \(2^{20}\) pages

d. What is the minimum number of bits required for the page table base address register?  
   20 bits

Everybody Got Choices

e. Answer "Yup!" (True) or "Nope!" (False) to the following questions
   i. The page table is stored in main memory  \(\text{Yup!}\)
   ii. Every virtual page is mapped to a physical page  \(\text{Nope!}\)
   iii. The TLB is checked before the page table  \(\text{Yup!}\)
   iv. The penalty for a page fault is about the same as the penalty for a cache miss  \(\text{Nope!}\)
   v. A linear page table takes up more memory as the process uses more memory  \(\text{Nope!}\)
Question 2:

F2) V/I/O)rtual Potpourri (23 pts, 30 mins)
For the following questions, assume the following:

- 16-bit virtual addresses
- 4 KiB page size
- 16 KiB of physical memory with LRU page replacement policy
- Fully associative TLB with 4 entries and an LRU replacement policy

a) What is the maximum number of virtual pages per process? ________________

16

b) How many bits wide is the page table base register? ________________

14

For questions (c) and (d), assume that:

- Only the code and the two arrays take up memory
- The arrays are both page-aligned (starts on page boundary)
- The arrays are the same size and do not overlap
- ALL of the code fits in a single page and this is the only process running

```c
void scale_n_copy(int32_t *base, int32_t *copy, uint32_t num_entries, int32_t scalar)
{
    for (uint32_t i=0; i < num_entries; i++)
        copy[i] = scalar * base[i];
}
```

c) If scale_n_copy were called on an array with \(N\) entries, where \(N\) is a multiple of the page size, how many page faults can occur in the worst-case scenario?

Answer: \(N/(2^9) + 1\)

d) In the best-case scenario, how many iterations of the loop can occur before a TLB miss?

Answer: \(2^{10}\)
Question 3:

For the following questions, assume the following (IEC prefixes are on your green sheet):

- You can ignore any accesses to instruction memory (code)
- 16 EIB virtual address space per process  // 2^64 B
- 256 MiB page size  // 2^28 B
- 4 GiB of physical address space  // 2^32 B
- Fully associative TLB with 5 entries and an LRU replacement policy
- All arrays of doubles are page-aligned (start on a page boundary) and do not overlap
- All arrays are of a size equivalent to some nonzero integer multiple of 256 MiB
- All structs are tightly packed (fields are stored contiguously)
- All accesses to structs and arrays go out to caches/memory (there is no optimization by reusing values loaded into registers)

```c
typedef struct { *double dbl; double fun;} doubleFun

void dbLCpy(doubleFun* measurer, double* dblsToCpy) {
    measurer->fun = 0;
    for (uint32_t i = 0; i < ARRAY_SIZE; i+=4) {
        measurer->dbl[i] += dblsToCpy[i];
        measurer->fun += dblsToCpy[i];
    }
    measurer->fun /= ARRAY_SIZE;
}
...

/* Now, the code goes on to call the function dbLCpy. Assume that space for the array pointed to by measurer->dbl was allocated at some time in the past and that all elements in the array were set to 0. The arrays dblsToCpy and measurer->dbl are each of length ARRAY_SIZE.*/
... // dbLCpy function call here
```

a) Fill the following table:

<table>
<thead>
<tr>
<th>Virtual Page Number Bits: 36</th>
<th>Virtual Address Offset Bits: 28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page Number Bits: 4</td>
<td>Physical Address Offset Bits: 28</td>
</tr>
</tbody>
</table>

b) Assume the TLB has just been flushed. What TLB hit to miss ratio would be encountered if sizeof(double) * ARRAY_SIZE = 256 MiB and we run the above code? Show your work.

6(2^3) TLB hit: 3 TLB misses

*Our loop has 2^n / (2^3 * 2^3) = 2^n iterations. For each iteration, there are 6 memory accesses. For the loop entirely, there are 6(2^n) total memory accesses.
*Before the loop, there is one memory access. After the loop there are two memory accesses (read & write measurer-> fun).
*Thus, there are 6(2^3)+3 total memory accesses. We require three pages, one for dblsToCpy and two for doubleFun all that miss initially when loading up. Thus, our hit-miss ratio is 6(2^3): 3.
c) In the best-case scenario, how many iterations can be executed with no TLB misses? Use IEC prefixes when reporting your answer. Show your work.

\[ 2^{16} \text{ iterations} = 16 \text{ MiB}. \]

In the best case scenario, all 5 slots in the TLB are full of valid info. We have 2 pages worth of measurer->dbl, 1 page for dbl->fun, and 2 pages for dblsToCpy. This means that our arrays are \(2^{16}\) B (2 pages) in length. We index by every 4 elements of 8-byte doubles each time to obtain \(2^{20} / 2^2 / 2^3 = 2^{16}\) iterations.

**Question 4:**

1. Hamming ECC

a) Given an N bit field, how many of those bits will be parity bits? \(\text{Ceil}(\log_2(N+1))\) is case it's correct

b) How long should a field be to store 15 bits of data with Hamming ECC for single error correction?

\[ 20: \quad 5 \text{ parity and 15 data} \quad \text{PPDPDDDPDDDDDDDDDDPDDD} \]

2) Assume that we have an encoded value, 1001110\text{two} with a single-bit error. Indicate below each parity bit if it has an error:

<table>
<thead>
<tr>
<th>Parity Bit</th>
<th>P1</th>
<th>P2</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK/ERROR</td>
<td>OK</td>
<td>Error</td>
<td>Error</td>
</tr>
</tbody>
</table>

Incorrect bit position: _6_

Correct data: _0100\text{two}_

2. RAID (Partially from the Su '12 Final)

1) Which type(s) of RAID (1, 3, 4 or 5) would be best to fit the following needs?

   a) Many fast reads _4, 5_

   b) Many fast writes _5_

   c) Fast reads of critical information _1_

   d) Fast reads of small, byte-size data _3_

2) There's a single disk failure in a disk array (you know which disk failed) and you want to read a single page. Assume that for block-striped arrays, the page is contained within a single block.

   a) What's the fewest number of disks you have to read from if the array were:

   i. RAID 1 with 2 disks _1_

   ii. RAID 5 with 4 disks _1_

   Data is NOT in the disk that failed

   b) (2 point) What's the greatest number of disks you have to read from if the array were:

   i. RAID 4 with 4 disks (including parity) _3_

   ii. RAID 5 with 4 disks _3_

   Data IS in the disk that failed and needs to be recovered
3a) When an exception occurs, the MIPS processor does all of the following except:

a. reads the Cause register  
b. runs the code starting at location 0x80000080  
c. switches to kernel mode and disables interrupts  
d. saves the address of the instruction that raised the exception

Perhaps the exception handler software will read the Cause register, but the processor doesn't do that on its own. (The processor will *set* the Cause register to a value indicating the reason for the exception.)

3b) The main advantage of using interrupts is:

a. allows the processor to do other useful tasks while waiting for slow I/O  
b. allows centralized error handling  
c. allows the processor to switch to kernel mode  
d. allows a user program to have access to I/O devices

The others may be advantages of having an operating system, but not specifically of interrupts.

How should the following devices share information with a computer?  
Pick between Polling, Interrupt

a) Mouse  
b) Hard drive  
c) Printer  
d) Network Cards  
e) Keyboard