SOLUTIONS to Problem 10
Computer Architecture

(a) (1 Point total) Team Alpha has come up with a CPU that has a clock frequency of 3GHz. All instructions take 1 clock cycle, except the floating point division that takes 13 cycles.

i. (0.5 points) Team Beta has come up with an alternative design that takes 1 cycle for all instructions except the floating point division that takes only 8 cycles. However, their CPU runs at a clock frequency of 2GHz. What percentage of the instructions in a program must be floating point divisions for Beta to show better performance than Alpha?

Let $0 \leq x \leq 1$ represent the percentage of floating point division instructions. For Beta to be better than Alpha, we must have:

$\text{Runtime}_{\text{Beta}} < \text{Runtime}_{\text{Alpha}}$
$1/2\text{GHz} \times (8 \times x + 1 \times (1- x)) < 1/3\text{GHz} \times (13 \times x + 1 \times (1- x))$
which results in $x > 1/3$, which means at least 33% of the instructions must be floating point division for Beta to outperform Alpha.

ii. (0.5 points) Team Gamma has a design similar to Team Alpha's design, except that their floating point division takes 11 cycles. What percentage of the instructions in a program must be floating point divisions for Beta to show better performance than Gamma?

Similar to the above case, but this time the equation becomes:

$\text{Runtime}_{\text{Beta}} < \text{Runtime}_{\text{Gamma}}$
$1/2\text{GHz} \times (8 \times x + 1 \times (1- x)) < 1/3\text{GHz} \times (11 \times x + 1 \times (1- x))$
which results in $x < -1$, which means no matter what percentage of instructions are floating point divisions, team Beta's design cannot beat Team Gamma's.

(b) (2 Points total) We have a 5-stage pipeline processor {IF, ID, EX, MEM, WB}. Consider the following piece of code:

```
I1: SW R16, 10 (R6)
I2: LW R4, 18 (R16)
I3: ADD R5, R4, R4 ; R5 is the destination
I4: OR R1, R2, R3
I5: AND R2, R1, R4
I6: OR R1, R1, R2
```

i. (0.6 points) Indicate data dependences and their types.

I1: SW R16, 10 (R6)	
I2: LW R4, 18 (R16)	RAW on R4 from I2 to I3 and I5 (3 inst below, though)

```
I3: ADD R5, R4, R4
I4: OR R1, R2, R3            RAW on R1 from I4 to I5 and I6
I5: AND R2, R1, R4           RAW on R2 from I5 to I6
I6: OR R1, R1, R2            WAR on R2 from I4 to I5
                             WAR on R1 from I5 to I6
                             WAW on R1 from I4 to I6
```

ii. (0.6 points) Assuming that no forwarding is implemented, insert NOP instructions to avoid hazards. How many instruction cycles does it take to run the above code from the IF stage of the first one to the WB stage of the last one?

```
SW R16, 10 (R6)
LW R4, 18 (R16)
NOP
NOP
ADD R5, R4, R4
OR R1, R2, R3
NOP
NOP
AND R2, R1, R4
NOP
NOP
OR R1, R1, R2
```

Overall takes 16 cycles.

iii. (0.6 points) Assuming that full forwarding is implemented, insert NOPs to eliminate hazards. Full forwarding includes ALU output to the EX stage of the next instruction without hazard, but loads cannot forward to the EX stage of the next instruction. How many instruction cycles does it take to run the above code from the IF stage of the first one to the WB stage of the last one?

```
SW R16, 10 (R6)
LW R4, 18 (R16)
NOP                  ;  delay I3 to avoid RAW hazard on R4 from I2
ADD R5, R4, R4       ; value for R4 is forwarded from I2 now
OR R1, R2, R3
AND R2, R1, R4       ; no RAW on R1 from the OR inst (forwarded)
OR R1, R1, R2        ; no RAW on R2 from the AND inst (forwarded).
                     ; no RAW on R1 from the OR inst above (forwarded).
```

Overall takes 11 cycles.

iv. (0.2 points) If implementing forwarding results in 20% increased cycle time, is it worth it for the code segment discussed in the previous parts?

Without forwarding, we have 16 * 1 time units to execute, but with forwarding we have 11 * 1.2 = 13.2 time units, so forwarding is worth it in this example.

(c) (1 point total) Consider a single-level virtual memory system in which the page size is 4KB, physical DRAM is 16GB and the virtual address is 43 bits. Each page table entry (PTE) also has bits for valid, dirty and protection. Round up PTE sizes to be multiples of bytes for easy indexing.

i. (0.5 points) How much physical memory is needed for storing the page table?

The virtual address is divided into 12 bits for the page offset and 31 bits for the virtual page address.

Virt page addr (31 bits)	Page offset 12

As a result, there must be 2^31 entries in the page table (2G entries)

The physical address is divided into 22 bits of physical page address and 12 bits of page offset.

Phy page addr (22 bits)	Page offset 12

PTEs need 22 bits for the physical page address, plus three bits for the flags, results in 25 bits, which is rounded up to 4 bytes. As a result, the page table takes 8GB of physical memory.

ii. (0.2 points) What is the disadvantage of using a single-level virtual memory system in our example?

It takes up too much space. In our example, half of the DRAM is used for the virtual table.

iii. (0.3 points) Suppose the architecture team has decided to change the TLB to harness a 2-level fully associative cache. A designer in the team makes the argument that they have to implement the tag search algorithm in hardware because the bottleneck in handling page faults is to find out if a page is missing from the physical memory. Is that argument valid? Explain.

No the argument is not valid. First of all, since accessing hard disk is orders of magnitude slower than accessing DRAMs, disk access is the bottleneck in virtual memory, and not the tag search process. In fact, the overhead of software tag search is insignificant compared to disk access times. As a result, it does not really matter much if we implement the tag search process in hardware or software.

Furthermore, a software implementation can use sophisticated algorithms and data structures for its page replacement policy to reduce miss rate. Such an implementation would be difficult in hardware. So contrary to the argument of the designer, software implementation of the tag search algorithm is indeed preferable to its hardware counterpart.