PhD Qualifier Exam, Fall 2017

Computer Architecture and Systems

1. (6 points) Short Answer Questions.
(a) (2 points) What is a *page table*? Explain why many modern computer systems use a hierarchical page table.

A page table is a data structure that maintains the current set of Virtual Page Number (VPN) to Physical Frame Number (PFN) mappings.

Modern computer systems have very large virtual address spaces, and therefore the number of virtual pages is very high. Most programs use only a tiny portion of this virtual address space. As the maximum number of entries in a page table is equal to the number of virtual pages, using a linear data structure that allocates space for all of the VPNs is quite wasteful. By using a hierarchical data structure, a process can start with a small-size page table and gradually build it as when more VPNs are used.

b) (2 points) Many modern computer systems have extremely large amounts of physical memory. Explain why many of them still implement the *virtual memory* concept.

Besides allowing a physical memory size that is smaller than the virtual address space, the virtual memory concept provides additional benefits related to multitasking, such as fast context switching, sharing, and protection.

The virtual memory concept makes it possible for virtual pages from multiple processes to simultaneously reside in the physical memory (in different page frames). Therefore, the virtual pages of a process need not be saved (and restored) while doing a context switch.

Sharing of the virtual pages of multiple processes can be easily achieved by mapping the shared pages to the same physical frame.

Information regarding the access privileges of each virtual page can be maintained in the corresponding entry in the page table, making it easy to enforce different access privileges for different virtual pages.

(c) (2 points) Cache memory and virtual memory have many similarities: both take advantage of *locality of references* to effectively implement the memory address space as a hierarchy. Explain clearly why they use very different levels of associative mapping (cache memory implementations use very low levels of associativity whereas virtual memory implementations use very high levels of associativity).

The average memory access time obtained when using a 2-level hierarchical memory system is given by:

$$T_{Mem} = T_{L1} + mT_{Miss} \quad (1)$$

where T_{L1} is the access time for the first level of the hierarchy, m is the fraction of times the request missed in the first level and had to be forwarded to the next level, and T_{Miss} is the time required to service the miss. In order to minimize T_{Mem}, it is important to reduce both of the terms on the right-hand side.

For cache memories, T_{L1} is of the order of 1 ns and T_{Miss} is of the order of 10-100 ns. Cache memory

designs therefore try to have an m value around 0.01 to 0.1. Very low levels of associativity are used to keep T_{L1} low.

For a virtual memory system, T_{L1} is of the order of 100 ns and T_{Miss} is of the order of 10s of ms. Because these two values differ by five orders of magnitude, m has to be made extremely small (of the order of 0.00001) to have low T_{Mem}. High levels of associativity are therefore used to achieve low m.

2. (7 points) A single-cycle unpipelined processor has a clock cycle of 50 ns. An 8-stage pipelined version of the same processor is built as follows:

(i) Fetch stage 1 :- 7 ns
(ii) Fetch stage 2 :- 8 ns
(iii) Decode/Register read stage :- 5 ns
(iv) ALU stage 1:- 7 ns
(v) ALU stage 2:- 7 ns
(vi) Memory access stage 1 :- 7 ns
(vii) Memory access stage 2 :- 8 ns
(viii) Writeback/End stage :- 4 ns

Each pipeline latch has a latency of 1 ns.
The branch comparison is done in the ALU1 stage and the branch target is computed in the ALU2 stage. Assume that there is a split instruction cache and data cache. Assume also that register writes are done in the first half of a clock cycle and register reads are done in the second half of the clock cycle. Assume that there is a perfect cache memory (with no misses), no data forwarding, no branch prediction, and no delayed branching.

(a) (1 point) What is the best-case clock cycle time possible for this pipelined processor?

In a pipelined datapath, the stage that has the longest delay determines the best-case clock speed. The best-case clock cycle time possible is therefore 8 ns + 1 ns = 9 ns.

Consider the code snippet given below:

```
      LOAD  R1, 0(R2)       # load from Memory into R1
      BEQ   R0, R1, L1      # branch to label L1 if R0 and R1 are equal
      AND   R1, R1, R4      # R1 <-- R1 + R4
      LOAD  R1, 0(R1)       # load from Memory into R1
      SUB   R2, R1, 10      # R2 <-- R1 + 10
L1:   ADD   R2, R2, 5       # R2 <-- R2 + 5
```

(b) (3 points) Draw a pipeline timing diagram showing how this code snippet would be executed in the basic pipeline. Assume that the branch (instruction `BEQ`) ended up being taken. How many clock cycles does it take to complete the execution of this code snippet?

```
Cycle 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
LOAD: F1 F2 DR A1 A2 M1 M2 W
BEQ:     F1 F2 D  -  -  -  R  A1 A2 M1 M2 W
AND         F1 F2 -  -  -  -  -  sq
ADD                                F1 F2 DR A1 A2 M1 M2 W
```

The BEQ instruction stalls until it can read the value obtained by the LOAD instruction. The AND instruction stalls from cycle 5 onwards when it becomes known that there is a pending branch instruction (BEQ) in the pipeline. It is eventually squashed in cycle 10 after the outcome of the BEQ instruction becomes known. The BEQ target is calculated in cycle 10, and ADD instruction starts in cycle 11. Altogether, this code snippet takes 18 clock cycles to complete.

(c) (1 point) What is the speedup possible with this pipeline (over the unpipelined processor)?

The single-cycle unpipelined processor takes 50 ns for each of the 3 instructions, requiring a total of 150 ns.
The pipelined processor takes 18×9 *ns = 162 ns.*
Speedup of pipelined processor over unpipelined processor $= \frac{\text{Exec Time of Unpipelined}}{\text{Exec Time of Pipelined}} = \frac{150}{160} = 0.926.$
The slowdown is because we have considered the pipeline fill time. Once the pipeline is in steady-state, running longer programs, it will have speedup.

(d) (2 points) Re-draw the pipeline timing diagram assuming full forwarding of data values. How many clock cycles does it now take to complete the execution of this code snippet?

```
Cycle 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
LOAD: F1 F2 DR A1 A2 M1 M2 W
BEQ:     F1 F2 D  -  -  -  A1 A2 M1 M2 W
AND         F1 F2 -  -  -  -  sq
ADD                           F1 F2 DR A1 A2 M1 M2 W
```

Because of data forwarding, the value fetched by the LOAD instruction can be made available to the BEQ instruction 1 cycle earlier. The code snippet now takes 17 clock cycles to complete.

3. (7 points) Short answer questions.
(a) (2 points) Explain why most computer systems have at least two modes of operation—*kernel mode* and *user mode*. When does the computer operate in each of these two modes?

Modern computer systems have two (or more) operation modes so that the operating system can support features such as virtual memory, protection, and multitasking.

When the CPU is in the Kernel mode, the executing program has unrestricted access to the underlying hardware. It can execute any instruction in the ISA and reference any memory address. The Kernel mode is used to run the trusted routines of the operating system.

When the CPU is in the User mode, the executing program has limited access to the underlying hardware. It can execute only User mode instructions in the ISA and reference only User mode memory address space.

(b) (2 points) What are the functions performed by a *linker* while compiling a program?

The linker takes one or more object files and combines them into a single executable file. The following functions are involved in doing this:

- *Resolve cross-references among multiple object modules*
- *Search library files to find library routines used by the object modules*
- *Stitch together multiple object modules and etermine the memory locations that will be occupied by the sections in each object module*
- *Append a start-up routine at the beginning of the program*

(c) (3 points) Consider a computer system that implements multitasking. Explain how the system is able to temporarily stop a process, run other processes, and eventually run the stopped process as if there was no interruption.

In a system that implements multitasking, an exception event such as device interrupt, trap instruction, error/fault causes the following actions to take place:

- *Switch to Kernel mode*
- *Save the current PC value*
- *Update PC with a* `.ktext` *address to transfer control to the OS*

A hardware timer is used to raise an interrupt when a User program has run a quantum of time so that the OS can take control. Once OS gains control of the CPU, it saves the interrupted process' register values (and other useful processor state) so that it can restore the register values prior to running the interrupted process again.

The memory values of the interrupted process cannot be saved (and restored) in the same way as is done for the register values. Instead, the virtual memory concept is used to keep frequently used portions of the virtual address space of multiple processes simultaneously in the physical memory.