Final Exam

600.233 Computer Systems Fundamentals
Fall 2016
Johns Hopkins University
Instructor: Prof. Philipp Koehn

16 December 2016

Complete all questions.

Use additional paper if needed.

Time: 60 minutes.

Name of student:

____

Q1. Ten Questions *25 points*

Answer the following questions either with YES or NO.

1. Does loading a value from a hard-coded memory address takes two instructions in MIPS?
   NO. 3: one to load the lower 2 bytes, one to load the higher 2 bytes into a register, and then one to load the value

2. Does MIPS use a stack for the return address of a jump to subroutine?
   NO: uses a register

3. Can a MIPS instruction carry out both a memory lookup and a subtraction?
   YES: EXE and MEM are in different steps, can modify memory address with subtraction before retrieving value

4. To reduce cache misses with twice the cache memory is it better to double block size than double the number of blocks?
   Double block size: due to data locality, likely neighboring block is being used

5. Is a joint component for instruction fetch and data fetch an example for a structural hazard?
   YES: this is avoided in MIPS with separate instruction cache and data cache

6. Do x86 processors follow the Complex Instruction Set Computer (CISC) design?
   YES

7. Does the linker change the addresses of machine code instructions?
   YES

8. Do dynamically linked shared libraries avoid adjustment of addresses because they are loaded always into the same memory location?
   NO: uses instruction point register and global offset table

9. Can you always avoid using the stack and use registers instead for return addresses in function calls?
   NO: recursive calls would overwrite registers

10. Is it possible to run a process as fast in a virtual machine as in real machine?
    YES: if there are no emulated device interaction, process runs the same way

11. Do you want 5 points?
    YES: Points are good, mmmkay?

Q2. Caching *25 points*

Consider use of a 2-way associative cache that addresses blocks of 4 bytes, with 4 sets in a 8-bit address space.

(a) How are the 8 bits of the address used as tag, index, and offset for the cache?

bit 0-1: offset
bit 2-3: index
bit 4-7: tag

(b) Consider a following sequence of requests to the cache.

Enter the **tag** for each cache slot after each request in the table below. Assume FIFO as caching strategy (do not worry about internal bookkeeping of timestamps). Note: use " to indicate that the value in the slot is identical to the previous value.

Request	**Set 0**		**Set 1**		**Set 2**		**Set 3**	
	Slot 0	**Slot 1**	**Slot 0**	**Slot 1**	**Slot 0**	**Slot 1**	**Slot 0**	**Slot 1**
	empty	empty	empty	empty	empty	empty	empty	empty
00110101			*0011*					
01101000			*0011*		*0110*			
01101001			*0011*		*0110*			
10010111			*0011*	*1001*	*0110*			
10010110			*0011*	*1001*	*0110*			
10110001	*1011*		*0011*	*1001*	*0110*			
10110101	*1011*		*1011*	*1001*	*0110*			

Q3. Pipelining *25 points*

Consider the following sequence of MIPS commands.

```
0000    load $s0, 20($t0)
0004    load $s1, 24($t0)
0008    sub $s2, $s0, $s1
000c    bne $s2, $zero, 0018     ; assume that the test fails
0010    add $s4, $s2, $s1
0014    add $s2, $s2, $s1
0018    jr $ra                   ; return from subroutine
```

(a) Identify a data hazard in the code.

0004 load → 0008 sub: value in $s1 needs to be loaded first

(b) Identify a control hazard in the code.

000c bne → 0010 add: next instruction may not be 0010

(c) Complete the table on the next page that shows which command is currently in which stage of the CPU.

Identify each command by its address and operation name. End with instruction fetch of the `jr` command. Add more lines / use less lines as needed.

State your assumptions about handling data hazards and control hazard.

Time	Instruction Fetch	Instr. Decode / Register Read	Execute / ALU	Memory Access	Write-Back
1	`0000 load`	-	-	-	-
2	*0004 load*	*0000 load*			
3	*(bubble)*	*0004 load*	*0000 load*		
4	*0008 sub*	*(bubble)*	*0004 load*	*0000 load*	
5	*000c bne*	*0008 sub*	*(bubble)*	*0004 load*	*0000 load*
6	*0010 add*	*000c bne*	*0008 sub*	*(bubble)*	*0004 load*
7	*0014 add*	*0010 add*	*000c bne*	*0008 sub*	*(bubble)*
8	*0018 jr*	*0014 add*	*0010 add*	*000c bne*	*0008 sub*
9		*0018 jr*	*0014 add*	*0010 add*	*000c bne*
10			*0018 jr*	*0014 add*	*0010 add*
11				*0018 jr*	*0014 add*
12					*0018 jr*
13					
14					
15					

Data hazard: can be detected from instructions

Control hazard: Assuming do-not-take-branch prediction

Q4. Virtual Memory *25 points*

(a) Process memory.

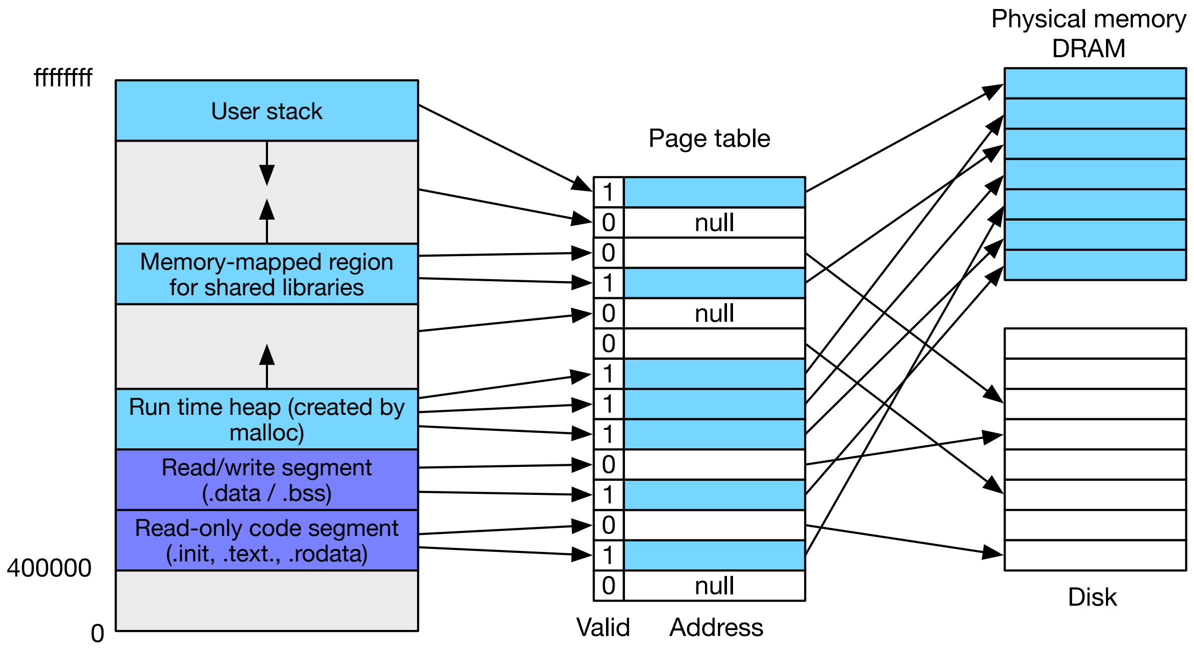
Draw a diagram that shows:

- virtual memory layout of a process,
- its page table,
- the system RAM,
- and the system disk

The diagram should include the following memory areas of a process:

- code
- shared libraries
- stack, and
- allocated memory for data structures.

Note: assume that the page table is just a flat lookup table and the RAM does not use any caches.


ffffffff
User stack
Memory-mapped region for shared libraries
Run time heap (created by malloc)
Read/write segment (.data / .bss)
Read-only code segment (.init, .text., .rodata)
400000
0
Page table
1
0 null
0
1
0 null
0
1
1
1
0
1
0
1
0 null
Valid
Address
Physical memory DRAM
Disk

(b) Change in memory use.

Indicate how entries in page table and memory use in RAM change ...

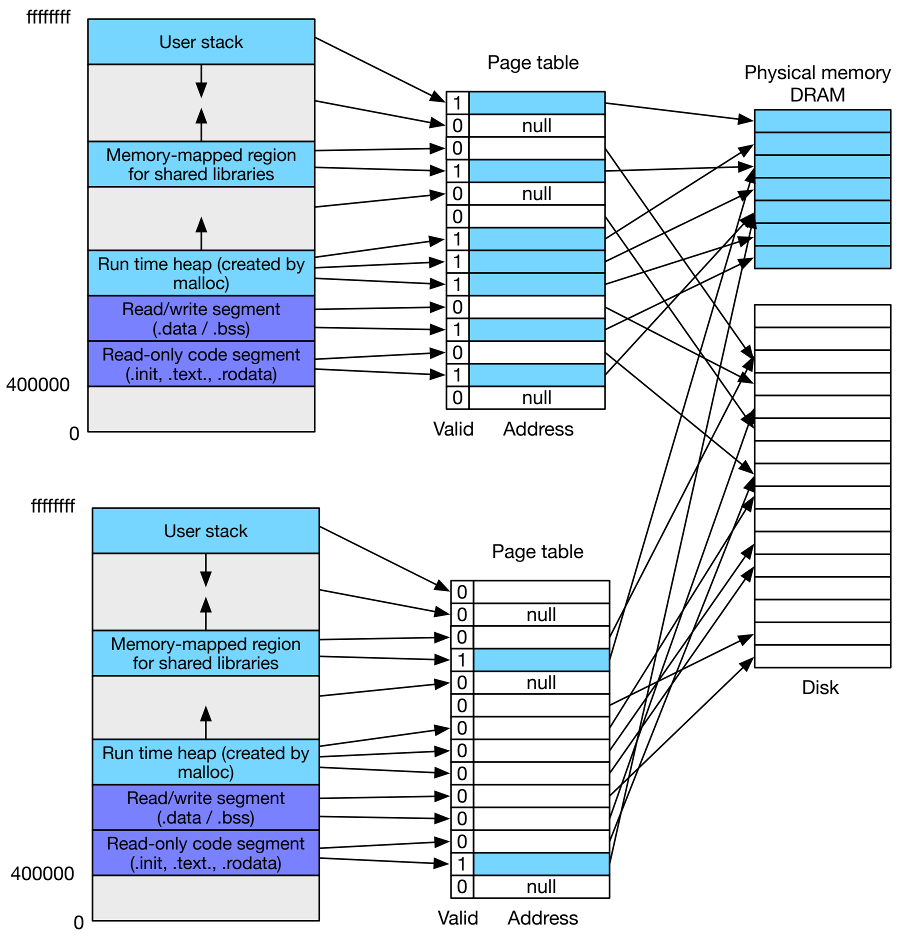
1. ... when process is launched
2. ... when process executed some instructions
3. ... when process allocates memory for a data structure

launched: page table is created
executed: code and data pages are loaded on demand
allocate memory: new page table entries are created, point to disk object in swap partition

(c) Multiple processes.

Now a second process using the same code is launched. Draw its page table and memory use and how it relates to the first process.

see next page


ffffffff
User stack
Memory-mapped region for shared libraries
Run time heap (created by malloc)
Read/write segment (.data / .bss)
Read-only code segment (.init, .text., .rodata)
400000
0
Page table
1
0 null
0
1
0 null
0
1
1
1
0
1
0
1
0 null
Valid Address
Physical memory DRAM
Disk
ffffffff
User stack
Memory-mapped region for shared libraries
Run time heap (created by malloc)
Read/write segment (.data / .bss)
Read-only code segment (.init, .text., .rodata)
400000
0
Page table
0
0 null
0
1
0 null
0
0
0
0
0
0
0
1
0 null
Valid Address

Read-only code and shared library point to the same locations in disk/RAM. After start-up, no other pages of the second process are loaded into RAM