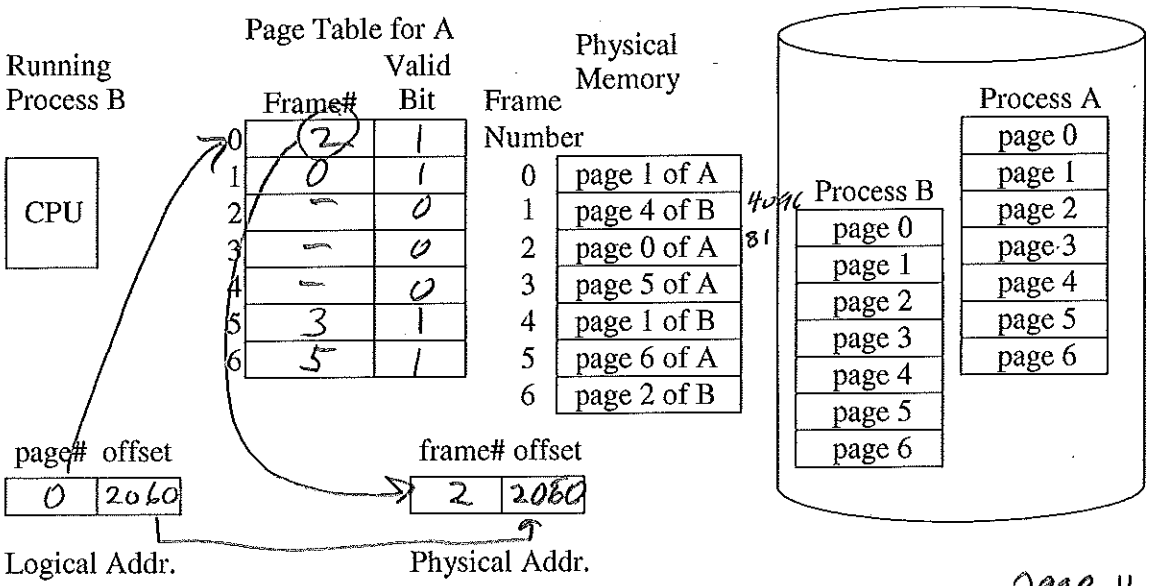
Mark F.

Comp. Arch. Homework #2 Due: Friday (9/20) 4 PM

1. Consider the demand paging system with 4096-byte pages.



a) Complete the above page table for **Process A**.

page # offset
0 | 2060

0.5 b) If process A is currently running and the CPU generates a logical/virtual address of 2060_{10}, then what would be the corresponding physical address? frame offset 2 | 2060 $2 \times 4096 + 2060 = 10,252_{10}$

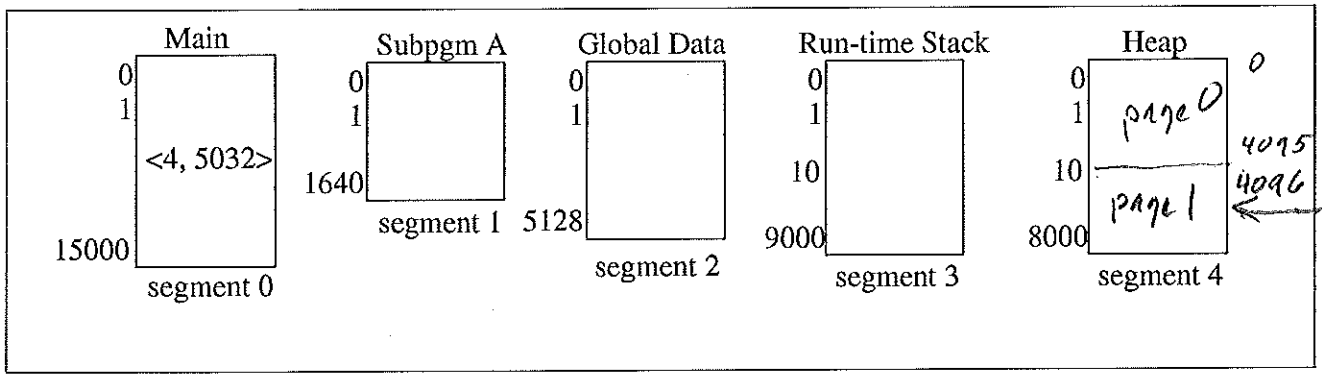
0.5 2. Explain how a TLB (translation-lookaside buffer) speeds the process of address translation?
TLB is a cache in CPU to hold recent page-table entries, so CPU typically avoids going to Page table

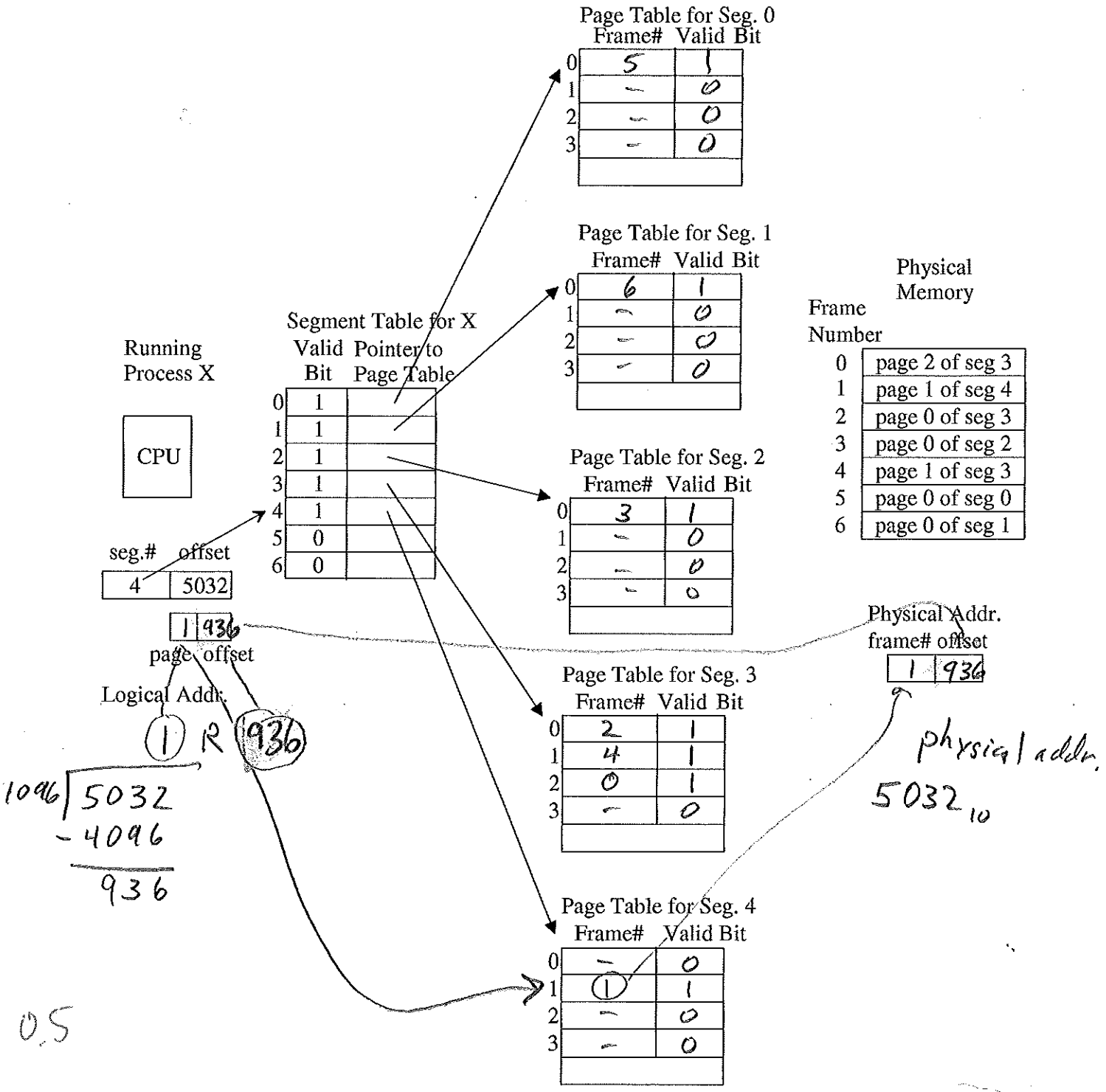
3. What advantages does combining paging and segmentation (i.e., paging of each segment) have over:

0.5 a) only paging allows protection of pgm's logical components along natural boundaries

b) only segmentation aids in memory management since pages fit in an memory frame and only needed parts of a segment are in memory.

4. Assuming a page size of 4096 bytes, complete the Page Tables for the pages in memory, and determine the physical address for the logical address <4, 5032>.




Page Table for Seg. 0
Frame# Valid Bit
0 5 1
1 - 0
2 - 0
3 - 0
Page Table for Seg. 1
Frame# Valid Bit
0 6 1
1 - 0
2 - 0
3 - 0
Page Table for Seg. 2
Frame# Valid Bit
0 3 1
1 - 0
2 - 0
3 - 0
Page Table for Seg. 3
Frame# Valid Bit
0 2 1
1 4 1
2 0 1
3 - 0
Page Table for Seg. 4
Frame# Valid Bit
0 - 0
1 1 1
2 - 0
3 - 0
Physical Memory
Frame Number
0 page 2 of seg 3
1 page 1 of seg 4
2 page 0 of seg 3
3 page 0 of seg 2
4 page 1 of seg 3
5 page 0 of seg 0
6 page 0 of seg 1
Running Process X
CPU
Segment Table for X
Valid Bit Pointer to Page Table
0 1
1 1
2 1
3 1
4 1
5 0
6 0
seg.# offset
4 5032
1 936
page offset
Logical Addr.
1 R 936
Physical Addr.
frame# offset
1 936
1096 | 5032
-4096
936
physical addr.
5032 10
0.5

5. (This question deals with the following toy virtual memory system which is tiny...) You have a byte-addressable memory with a two-entry TLB (fully-associate with Page as the tag), a 2-way set-associative cache (Tag bits shown in binary), and a page-table for a process P. Assume cache blocks of 8 bytes and page size of 16 bytes, so two blocks per frame. In the system below, main memory is divided into blocks, where each block's content is represented abstractly by a letter.

TLB

Page	Frame
0	3
4	1

Cache

	Tag	Data	Tag	Data
Set 0	00_2	C	01_2	I
Set 1	00_2	D	10_2	H

Page Table for P

	Frame	Valid Bit
0	3	1
1	0	1
2	-	0
3	2	1
4	1	1
5	-	0
6	-	0
7	-	0

Main Memory

Frame		Block
0	C	0
	D	1
1	I	2
	J	3
2	G	4
	H	5
3	A	6
	B	7

$8 = 2^3$ blocks
$8 = 2^3$ bytes/block
2^6 memory size

Virtual Memory for Process P

Page		Block
0	A	0
	B	1
1	C	2
	D	3
2	E	4
	F	5
3	G	6
	H	7
4	I	8
	J	9
5	K	10
	L	11
6	M	12
	N	13
7	O	14
	P	15

pages
$8 = 2^3$

page size
$16 = 2^4$

Given the system state as depicted above, answer the following questions:

a) How many bits are in a virtual address for process P? 7-bits
b) How many bits are in a physical address? 6-bits
c) Show the address format for a logical/virtual address including field names and number of bits. 3-bits 4-bit | Page # | offset |
d) Using your format in part (c), convert the virtual address 18_{10} to binary and put it in the appropriate fields. Now, explain how these fields are used to translate to the corresponding physical address.
e) Show the address format for a physical address including field names and number of bits that are used to check the cache.
f) Given that virtual address 6_{10} translates to physical address 54_{10}. Using your format in part (e), convert the physical address 54_{10} to binary and put it in the appropriate fields. Now, explain how these fields are used to locate physical address 54 in the cache.
g) Given that virtual address 25_{10} is located on virtual page 1, offset 9. Indicate exactly how this address would be translated to its corresponding physical address and how the data would be accessed. Include in your explanation how the TLB, the page table, cache, and memory are used.

Question #5 HW#2

.25 a) 7-bits $\Rightarrow$ 2^3 pages $\times$ 2^4 bytes/page $= 2^7$ virtual memory size

.25 b) 6-bits $\Rightarrow$ 2^3 blocks $\times$ 2^3 bytes/block $= 2^6$ physical memory

.25 c)

3-bit	4-bit
page #	offset in page

.25 d) 18_{10} = 64 32 16 | 8 4 2 1

0 0 1 | 0 0 1 0_2

4 2 1

page 1 offset 2

In general, we check page table entry 1 and find page 1 in frame 0. offset just copied.

frame#	offset
0 0	0 0 1 0

$\Rightarrow$ physical addr. 2_{10}

.25 e) Physical address has two views

2-bits	4-bits
frame#	offset in page

and for checking cache

2-bits	1-bit	3-bits
tag	set #	offset in block

f) 54_{10} = 32 16 | 8 4 2 | 1

1 1 | 0 | 1 1 0

tag = block #; set 0; offset 6 into block

(1) set 0 checked for tag 11_2

(2) No match, so a miss block 6 loading into cache set 0

(3) now, cache can supply offset 6 of block 6 loaded from memory.

Q5 (continued)

9) 25_{10} =

64	32	16	8	4	2	1
0	0	1	1	0	0	1

page 1_{10} | offset into page 9_{10}

Steps:

(1) CPU generates virtual addr. 25_{10} and splits it into page 1 and offset 9_{10}.

(2) TLB checked for page table entry for page 1, but its not there -- a miss in TLB.

(3) Load page table entry 1 from page table into TLB for future use

.25

(4) Build physical address

frame	offset 9
00	1001

(5) Re-interperate bits as

tag:	set #	offset in block
00	1	001

(6) Set 1 checked for tag 00_2 in cache.

(7) Hit, so cache can supply byte $001_2 = 1_{10}$ from the start of the block "0"

Note: The location of the page table is vague in this problem, but it would be in main memory or on disk if multi-level page table is used.

3

(5)