Department of Electrical and Computer Engineering

The University of Texas at Austin

EE 460N, Fall 2014
Lab Assignment 5
Due: Sunday, November 16 2014, 11:59 pm
Yale Patt, Instructor
Stephen Pruett, Emily Bragg, Siavash Zangeneh, TAs

Introduction

The goal of this lab assignment is to extend the LC-3b simulator you wrote in Lab 4 to handle virtual memory. You will augment the existing LC-3b microarchitecture in order to support virtual to physical address translation. You will also provide microarchitectural support for page fault exceptions and change the protection exception from Lab 4 to be based on access mode in the PTE.

Specification

Virtual memory

The virtual address space of the LC-3b is divided into pages of size 512 bytes. The LC-3b virtual address space has 128 pages, while physical memory has 32 frames. The LC-3b translates virtual addresses to physical addresses using a one-level translation scheme. Virtual pages 0–23 comprise the system space. They are mapped directly to frames 0–23 and are always resident in physical memory. The system space may be accessed with any instruction in supervisor mode, but only with a TRAP instruction in user mode. The remaining virtual pages (24–127) comprise the user space and are mapped to frames 24–31 via a page table stored in system space.

The page table contains PTEs for *both* the system and user space pages. It resides at the beginning of frame 8 of physical memory. A page table entry (PTE) contains only 9 bits of information but, for convenience, is represented by a full 16 bit word. Thus one PTE occupies two memory locations. The format of each PTE is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	PFN					0	0	0	0	0	P	V	M	R

If the protection (P) bit is cleared, the page is protected: it can only be accessed in supervisor

mode or by a TRAP instruction. Otherwise, the page can be accessed in either user or supervisor mode. The valid (V) bit indicates whether the page is mapped to a frame in physical memory (V = 1) or not (V = 0). The modified (M) bit indicates whether the page has been written to since it was brought in (M = 1) or not (M = 0). The reference (R) bit is set on every access to the page and cleared every timer interrupt.

Note that the PFN, the modified, and the reference bits are meaningless if the page is not resident in memory (the valid bit is zero).

Exceptions

Lab 5 supports four exceptions, listed below in order of priority (highest to lowest):

Exception	Vector	Occurs when …
Unaligned access	`x03`	a word-sized memory access to an odd virtual address is attempted
Protection	`x04`	the machine is in user mode and a protected page is accessed by any instruction other than the TRAP instruction
Page fault	`x02`	the page accessed by a user program is not valid (not in physical memory)
Unknown Opcode	`x05`	the program tries to use an unknown opcode (1010 or 1011)

Note that unlike Lab 4, in Lab 5 the unaligned access exception has a higher priority than a protection exception.

All three memory exceptions are detected immediately prior to the exception generating memory access. Therefore, when a control instruction loads the PC with an invalid address, the machine does not initiate the exception until it attempts to fetch from that invalid address.

Implementation

Additions to Datapath

During the execution of instructions, your microcode will convert virtual addresses to physical addresses and modify PTEs when necessary. To make address translation possible, we have added two registers to the datapath: Page Table Base Register (PTBR) and Virtual Address Register (VA). The PTBR points to the first entry of the page table in physical memory and will be initialized by the shell code to the starting address of the page table file you provide as a command line parameter to the simulator. To put the PTBR register onto the bus, you should assert the GatePTBR signal. The VA register is a temporary register to hold the current

address being translated. To put the VA register onto the bus and to write to the VA register from the bus, you should assert the GateVA and LD.VA control signals, respectively.

Address Translation

We will use a simple one-level translation scheme to translate virtual addresses to physical addresses. The high 7 bits of the virtual address (bits[15:9]) specify the page number, and the low 9 bits of the virtual address (bits[8:0]) specify the offset within the page. All 7 bits of the page number will be used during translation. Assume that at the beginning of each address translation phase the virtual address is located in the MAR, and if the operation is a write, a source register holds the data to be written. Address translation consists of the following steps:

1. Save the MAR into the temporary register VA
2. Load the MAR with the address of the PTE of the page containing the VA:
    - MAR[15:8] ← PTBR[15:8]
    - MAR[7:0] ← LSHF(VA[15:9], 1)
3. i.e., **MAR ← PTBR + (2 × page_number)**.
4. Read the PTE of the page containing the VA into the MDR
5. Check for a protection exception
6. Check for a page fault
7. Set the reference bit of the PTE
8. If the pending access is a write, set the modified bit of the PTE
9. Write the PTE back to memory
10. Load the physical address into the MAR:
    - MAR[13:9] ← PFN
    - MAR[8:0] ← VA[8:0]
11. If the operation is a write, load the MDR with the source register

To add support for virtual memory, you first need to determine when you need to perform address translation. Then, you will need to determine how to add an address translation state sequence to the state diagram. You will also need to determine how to return back to the correct state once address translation is complete. For this, you will need to augment the microsequencer. You are free to add new control signals, gates, muxes, temporary registers as you wish as long as you fully document your changes.

Shell Code

A modified shell code has been written for you: **lc3bsim5.c**
You will need to copy and paste the code you wrote for Lab 4 into this new shell code. The shell code takes a new command line parameter that specifies the file containg the page table to be loaded into physical memory. To run the simulator, type:

```
lc3bsim5 <micro_code_file> <page table file> <program_file_1>
<program_file_2> ...
```

The first parameter is the microcode file as before. The second parameter is the page table in LC-3b machine language. Note that since the page table is in physical memory, the first line of this file should be a physical address. For all the program files (including the interrupt and exception handlers), which are in virtual memory, the first line should be a virtual address.

Writing Code

You will modify your source code and control store from Lab 4 to implement the specification above. Do not remove Lab 4 functionality, even if it's not mentioned in the specification (for example, the timer interrupt). You will also create the following files to construct a particular execution scenario:

- Page table (starting at physical address `x1000`). The page table will be initialized upon simulator start-up based on the contents of the page table file you provided as a command line parameter to the simulator. Map pages 0–23 (system space) to frames 0–23 and mark them as valid and protected. Map pages 24, 96, and 126 (the user stack) to frames 25, 28, and 29, respectively, and mark them as valid and unprotected. Mark all other pages as invalid. All invalid PTEs for pages in user space must be marked as unprotected. Why?
- Other files are loaded into virtual memory and are described in the table below:

File	StartingVA	Description
User program	`x3000`	This program should calculate the sum of the first 20 bytes stored in memory starting at address `xC000`. This sum should then be stored as a 16-bit word at `xC014`. Then the program should jump to the address pointed to by this sum.
User program data	`xC000`	Use the following data: `x12`, `x11`, `x39`, `x23`, `x02`, `xF6`, `x12`, `x23`, `x56`, `x89`, `xBC`, `xEF`, `x00`, `x01`, `x02`, `x03`, `x04`, `x05`, `x06`, `x07`
Interrupt/Exception vector table	`x0200`	Form the contents of this table based on the vectors of each interrupt/exception and starting addresses of the service routines for each interrupt/exception.
Timer interrupt	`x1200`	The interrupt service routine must traverse the entire page table, clearing the reference bits of

service routine		each PTE. You may assume when writing this code that the start address of the page table is fixed.
Page fault exception handler	`x1400`	All four should simply halt the machine
Protection exception handler	`x1600`	
Unaligned access exception handler	`x1A00`	
Unknown opcode exception handler	`x1C00`	

Note: you should also test your design with other execution scenarios.

What to Submit

The following table lists the files you need to submit:

File Name	Description
`readme`	A detailed explanation of your design. It should include: ● Changes you made to the state diagram. Include a picture similar to the state machine that shows the new states you added (only show your changes or mark your changes in a new state diagram). This picture should include the encodings of new states you added. Clearly show where each state fits in the current state diagram. Describe what happens in each new state. ● Changes you made to the datapath. Clearly show the new structures added, along with the control signals controlling those structures. Describe the purpose of each structure. ● New control signals you added to each microinstruction. Briefly explain what each control signal is used for. ● Changes you made to the microsequencer. Draw a

	logic diagram of your new microsequencer and describe why you made the changes. ● *Please upload your documentation on Canvas in PDF format before class on Monday, November 17, 2014.*
lc3bsim5.c	Adequately documented source code of your simulator
ucode5	Control store file used by your simulator
dumpsim	To generate this file, set up the execution scenario described above. Run the user program and dump the memory locations containing the page table once before the 300th cycle, once after the ISR is done, and finally after the protection exception halts the execution of the program (you should get a protection exception after the jump). Also, dump memory location `x3814`(corresponding to which virtual address?) and the current registers after the protection exception.
pagetable.asm	Page table
vector_table.asm	Interrupt and exception vector table
add.asm	User program
data.asm	Data for the user program
int.asm	Timer interrupt service routine

Note: all asm files must be LC3b assembly source code files, NOT hex or obj files.

Things To Consider

1. The supervisor stack should start at address `x2FFF` (SSP is initialized in the shell code to `x3000`). If the user program uses the user stack, the user stack should start at address `xFDFF` (i.e. the user program should initialize R6 to `xFE00`).
2. When do you need to do address translation? How will you return from the address translation to the correct state? An obvious solution would be to have a register which holds the 6-bit return address. Although this is acceptable, think about the possibility of other solutions, e.g. a mechanism that is similar to COND bits in the microsequencer?
3. Since the page table is in physical memory, and the PTBR contains a physical address, the interrupt service routine needs to access the PTEs using physical addresses. An easy way to do this is to turn off virtual to physical address translation when operating in system mode. This will have the added benefit of reducing the

number of cycles taken to execute the ISR. What restriction does turning off virtual to physical address translation place on the interrupt service routine code?

4. The microcode used for starting the exception service routine is very similar to the microcode that is used for starting the interrupt service routine. Are there any differences? With these differences in mind perhaps you can combine the states used for interrupt and exception handling.
5. You do not need to worry about nested interrupts or exceptions for this assignment.
6. If you were unable to finish Lab 4 and are hence starting from Lab 3, please make a note of this in the readme file. You will be unable to simulate the page fault exception, protection exception and timer interrupt if the requirements of Lab 4 are not done.