Heavenly Medina and Joshua Grabenstein
CSC 345-01
Project 3 - report.pdf
April 14, 2019

**Program Requirements (70 pts):**

**• (10 points) Correctly read in input logical addresses.**
The logical address is correctly read in from addresses.txt from the following code segment below:

```
  unsigned int laddr;             //logical address read in from addresses.txt
...
   char* readIn = argv[1];                  //reads in addresses.txt from command line to be read
   addressFile = fopen(readIn, "rt");

 while((fscanf(addressFile, "%d", &laddr)) == 1){
      laddr = laddr & 65535;
…
}
```

It opens the file as entered by the user in the command line ./main addresses.txt using argv[i]. The file is then opened to read (“rt”) and it is read into the unsigned int variable laddr declared earlier in the program.

**• (10 points) Correctly translated the input addresses into physical addresses.**

```
//masks to get page number and offset
#define PAGE_NUMBER_MASK 0x0000FFFF
#define OFFSET_MASK 0x000000FF
…
      page_number = (laddr & PAGE_NUMBER_MASK) >> 8;
      page_offset = laddr & OFFSET_MASK;
...
paddr = pageTable[page_number] *256 + page_offset;
```

First, the logical address is read in and the page number and offset number are obtained through masking the logical address.  Once these elements are obtained the physical address can be

determined by multiplying the element in the page table at the page number multiplied by 256 and then adding the offset.

**• (10 points) Correctly retrieved the values stored in the physical addresses.**

```
value = PhyMem[paddr];
   //the signed byte value stored in physical memory at the translated physical address
      fprintf(output3, "%d\n", value);
```

The values stored in the physical address are correctly retrieved and outputted into out3.txt by viewing the value stored in the physical memory at the physical address.

**• (10 points) Implemented FIFO based TLB update.**

```
        if(TLB_size == 16){
           TLB_size--;
        }

        for(TLB_index = TLB_size; TLB_index > 0; TLB_index--) {
           TLBPageNum[TLB_index] = TLBPageNum[TLB_index - 1];
           TLBFrameNum[TLB_index] = TLBFrameNum[TLB_index - 1];
        }
        if(TLB_size <= 15){
           TLB_size++;
        }
```

The TLB was implemented using a FIFO based update. The size of the TLB is kept tracked by a variable called TLB_size and it is either incremented or decremented determining on

**• (10 points) Correctly counted number of page faults**

```
      }else if(pageTable[page_number] == -1){
     ...
        page_faults++;
}
```

The page faults are counted by a variable called page_faults, and it is incremented when the value at pageTable[page_number] is equal to -1, signifying that the value at the page table is invalid because initially all variables in the page table are initialized to -1, so it is considered a page fault.

**• (10 points) Correctly counted number of TLB hits.**

```
 if(!(TLB_hit == -1)){
        TLB_hits++;
```

```
    }
```
The TLB hits are counted when the value TLB_hit variable is equal to -1, meaning that the TLB at that location is invalid and signifies a hit.

**• (10 points) Implemented FIFO based page replacement algorithm. (main_pr)**

```
else if(pageTable[page_number][0] == -1)//Page in table at given page number is empty
        {
                if(l != 0){//Memory is full, so FIFO page replacement
                        //First we need to invalidate the page in the page table that is currently holding the frame in memory we have to replace
                        temp = frameNum + ((k) * l);//Current frame plus 127 being a single version of memory, times the number of versions we are on.
                        temp2 = pgnumarr[temp];//Page number that we need to invalidate
                        pageTable[temp2][1] = 0;//Invalidates page.

                        //Now we need to load the requested page into the frame
                        fseek(backingStore, page_number*256, SEEK_SET);
                        fread(buf, sizeof(char), 256, backingStore);

                        for(int lpcnt = 0; lpcnt < 256; lpcnt++){
                                temp = (frameNum * 256) + lpcnt;
                                PhyMem[temp] = buf[lpcnt];
                        }

                        //Now we can store the frame number into the page table and validate the page. Then, calculate physical address and iterate page faults and frame number
                        pageTable[page_number][0] = frameNum;
                        pageTable[page_number][1] = 1;
                        paddr = pageTable[page_number][0] * PAGE_SIZE + page_offset;

                        frameNum++;
                        page_faults++;
                }

                else{//Memory isn't full, so add page to page table and insert page into frame
                        //First, load page into memory
                        fseek(backingStore, page_number*256, SEEK_SET);
                        fread(buf, sizeof(char), 256, backingStore);
```

```
                        for(int lpcnt = 0; lpcnt < 256; lpcnt++){
                                temp = (frameNum * 256) + lpcnt;
                                PhyMem[temp] = buf[lpcnt];
                        }
                        //Then, load frame number into page table. Validate validity bit, then, calculate physical address and iterate page faults and frame number
                        pageTable[page_number][0] = frameNum;
                        pageTable[page_number][1] = 1;
                        paddr = pageTable[page_number][0] * PAGE_SIZE + page_offset;

                        frameNum++;
                page_faults++;
                 }

                 if(frameNum > 127)//If frame number exceeds maximum possible frame in physical memory
                 {
                        k = 128;
                        frameNum = 0;
                        l++;
                }
        }

        else if(pageTable[page_number][0]!=-1)//If the page in the page table isn't empty
        {
                if(pageTable[page_number][1] == 0){//If the page in the page table is invalid
                        //First we need to load a new frame into memory
                        fseek(backingStore, page_number*256, SEEK_SET);
                        fread(buf, sizeof(char), 256, backingStore);

                        for(int lpcnt = 0; lpcnt < 256; lpcnt++){
                                temp = (frameNum * 256) + lpcnt;
                                PhyMem[temp] = buf[lpcnt];
                        }
                        //Then, load that frame number into page table
                        pageTable[page_number][0] = frameNum;
                        pageTable[page_number][1] = 1;//And make the page valid
                        paddr = pageTable[page_number][0] * PAGE_SIZE + page_offset;//Recalculate paddress
```

```
                frameNum++;
                page_faults++;
                }

        else if(pageTable[page_number][1] == 1){//Else the page is valid, so we retrieve the frame out of it and use it to calculate the physical address for the data we want to retrieve. This is the only outcome which results in a page hit.

                        //First we retrieve the frame # from the page table
                        temp = pageTable[page_number][0];//Temp is the frame number

                        //Now we recalculate the frame number
                        paddr = temp * PAGE_SIZE + page_offset;
                }

        if(frameNum > 127)//If frame number exceeds maximum possible frame in physical memory

                {
                        k = 128;
                        frameNum = 0;
                        l++;
                }
        }
```