LABORATORY MANUAL

MICROPROCESSORS
&
SYSTEMS DESIGN
B.E. (Electronics)
Semester VII

Index

Laboratory Session 1

Title: To write a program using C /C++ for simulating the Branch Prediction Logic of the Pentium processor.

References: The Pentium Processor – Tom Shanley

Pre-Requisites: Knowledge of C /C++

Theory:

The Pentium Processor includes the branch prediction logic allowing it to avoid pipeline stalls if it correctly predicts whether or not the branch instruction is executed. When a branch operation is correctly predicted none of the performance penalty is incurred. However, whenever the branch prediction is not correct, a three cycle penalty is incurred if the branch executed is in the U pipeline and four cycle penalty if it is executed in the V pipeline. The branch prediction mechanism is implemented using a four way set associative cache with 256 entries called the Branch target buffer (BTB).

Algorithm:

1. Start.
2. Initially no history is exist hence the branch is predicted as not taken.
3. Take user input, whether the branch is actually taken or not.
4. If initially the user decides that the branch is not taken, simply wait and continue in loop till the branch is actually taken.
5. Update history bits to 'Strongly taken'.
6. Predict whether the branch will be taken depending on he history bits.
7. If the prediction and the user input do not match, take necessary action. i.e. upgrade history when the branch is actually taken and down grade if not taken one step at a time in the following sequence.

   Strongly taken weakly taken weakly not taken strongly not taken

8. Go to step 3.

Post –Lab Assignments:

1. Explain the organization of the Branch target buffer (BTB).
2. Explain in detail how the branch prediction logic reduces pipeline stalling in the Pentium processor

Laboratory Session 2

Title: To write a program to simulate the MESI cache consistency model implemented in the Pentium processor.

References: The Pentium Processor – Tom Shanley

Pre-Requisites: Knowledge of C / C++ or MATLAB programming

Theory: The MESI cache consistency model provide a method of tracking the various states that a cache line can be stored in so as to ensure consistency across all possible sources of a given line without invalidating data stored in the caches. The four states in the MESI model are:

Modified: The line in the cache is stored as modified when it has been updated due to a write hit in the cache. This alerts the cache sub system to snoop the system bus and write the modified back to the memory when a snoop hit to the line is detected.

Exclusive: Indicates that the cache knows that no other cache in the system possesses a copy of this line, therefore it is exclusive to this cache.

Shared: Indicates that the line may be present in several caches and if so, an exact duplicate of this information exists in each source.

Invalid: The initial state after reset, indicating that the line is not present in the cache.

Every line in the cache is assigned one of these state indicators to identify the status of the information stored in the cache. Transitions from one state to another may be caused by a local processor read or write operation or a bus snoop when another bus master initiates bus activity.

Algorithm:

1. Start
2. Ask the user t input one of the following states i.e. 1 for invalid, 2 for shared, 3 for Modified and 4 for Exclusive.
3. If state inputted is 1 then

Check if WB/WT =0, if yes the change the state to shared else change the state to Exclusive.

4. If state entered is 2 then
   a. Check if it is write hit, if yes remain the same state.
   b. Else check if it is read hit, if yes then remain the same state.
   c. Else check if it is external snoop hit (INV=0) then remain the same state.

d. Else check if it is a external snoop hit on write (INV=1), if yes change the state to invalid.

e. Else check if it is a write hit and WB/WT# =1, if yes then change the state to exclusive.

5. If the state entered by the user is 3 then.
   a. Check if it is read a read hit, if yes then remain in the same state.
   b. Else check if it is a write hit, if yes the remain in the same state.
   c. Else check if it is INVD, if yes then change the state to Invalid.
   d. Else check if it is external snoop hit on write then change the state to invalid.
   e. Else check if it is external snoop hit on read (INV=0) then change the state to Shared.
6. If state entered by the user is 4. then
   a. Check if it is a Read hit, if yes, then remain the same state.
   b. Else check if it is a External Snoop hit on Read (INV=0), if yes, then change the state to Shared.
   c. Else check if it is write hit, if yes, then change the state to Modified.
   d. Else check if it is a Internal Snoop hit or External Snoop hit on Write (INV =1), if yes, then change the state to Invalid.
7. Output the current state.
8. Stop

Post Lab Assignments:

1. Explain the Write –Once policy of the Pentium Processor.
2. Explain the need for the MESI cache consistency model.

Laboratory Session 3

Title: Mouse driver using DOS interrupts.

References: MS-DOS by Ray Duncan

Pre-Requisites: 1. Knowledge of various DOS interrupts
2. Knowledge of Assembly Language.

Theory:

Device drivers are software codes which provide interface between the processor and the actual hardware. It does the function of initializing the device, securing it and detaching the device after its use. All operating systems provide built- in functions that can be used to write device drives. Thus function takes of the address and hardware requirements of the devices. Microsoft Mouse driver provides many functions for using the mouse which can be accessed using INT 33H DOS interrupt .Some of these functions are shown for eg. Pointers getting button status and pointer position etc.

Function 01 H: shows mouse pointer. Displays the mouse pointer and cancels any mouse pointers exclusion and previously designed by INT 33 H. Function 01 H call with AX = 0001H. Returns nothing.

Function 03 H: Get mouse position and pointer position. Returns current mouse button status and pointer position.

Call with AX 0003H
Returns: BX= Mouse button status
Bit significant (if set)
0 - left button is down
1 - Right button is down
2 – Centre button is down
3 – 15 reserved (0)
CX - horizontal X co-ordinate
DX - vertical Y co- ordinate Similar to INT 33 H, Microsoft provides functions for using the video display using INT 21 H.

INT 21H function 02H : character outputs the character to the currently active video display.
Call with AH = 02 H
DL = 8 bit data for output
Returns: Nothing
e.g. Send the character " * " to the standard
output device

```
MOV AH, 02
MOV DL,'*'
INT 21H
```

Algorithm:

1.Make mouse pointer visible using INT 33H function 01h.
2. Cut button status and pointer position using INT 33h function 03h.
3. Convert X & Y co-ordinate value into ASCII.
4. Display values of X & Y on the screen using INT 21h function 01h.
5. If button status is zero display “left click” on the screen of button status. & if it is one display “right click” on the screen.
6. Go to step two.

Post Lab Assignments:

1.Explain the principle of working of the Mouse

2.What is the procedure for loading any device driver in a system

Laboratory Session 4

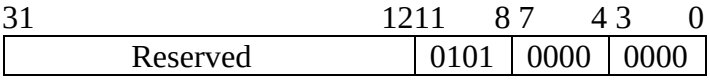
Title: To write an assembly language program (using Turbo Assembler) to study the CPUID instruction f the Pentium Processor.

References: The Pentium Processor – Tom Shanley

Pre-Requisites: Assembly Language programming for the Pentium (using TASM).

Theory: The CPU identification instruction provides with information regarding the vendor, family, model and stepping of the processor. The programmer uses a value in the EAX register to identify the information desired.
The defined EAX input values are

- 0000 0000h in EAX returns 0000 0001h in EAX for the Pentium processor and the ASCII representation of the string " Genuine Intel" is returned in EBX:EDX: ECX.
- 0000 0001h in EAX returns the information as shown below in the EAX register.



Family Model Stepping

EBX and ECX are set to zero.3fh is returned in the EDX register indicating feature supported by the processor. Feature bit zero is the only one currently defined; a one returned in this bit indicates that the FPU is present on –chip.

- If any other input value is used in EAX with the CPUID instruction EAX ,EBX , ECX and EDX are all returned with zeroes.

Algorithm:

1. Start.
2. Move 00h in EAX register.
3. Run the CPUID instruction.
4. Display the contents of EBX: EDX: ECX onto the display screen.
5. Move 0000 0001h in EAX register.
6. Run the instruction again.
7. Convert the obtained data in EAX to actual form to display on the screen.
8. Stop.

Post- lab Assignments:

1 How can the programmer determine if the processor supports the CPUID instruction ?
2. List out the other new instructions in the Pentium.

Laboratory session 5

Title : To introduce BIOS & DOS interrupt service routine to write assembly language program for pixel based graphics.

References:

1. The 80386DX microprocessor - Walter Triebel
2. 80386 programming - Pappas Murray
3. MS-DOS by Ray Duncan

Pre- requisites:

1. Knowledge of Assembly language programming
2. Knowledge of DOS interrupts.

Theory:

The BIOS function request in this category are used to control graphics on the PC monitor. The function request is choosing by setting the AH register to the Appropriate value and using interrupt 10H.

Function 0FH
Get current video mode
Returns current mode no in AL

Function 00H
Set video mode.
AL: mode no to indicate deserved mode
Returns nothing

```
e.g.  MOV AL,12      }
      MOV AH,00H     } 640X480 graphics
      INT 10H        }
```

Function 0CH
Write pixel dot
AL: colour e.g. 02 for green
CX: Column Number
DX: Row number
BH: Page number

Algorithm:

1. Get current video mode and save
2. Set 640X480 video mode.
3. Load AL with 02, CX & DX with appropriate column and row number.
4. Call function 0CH to write a pixel dot.
5. Increment CX i.e. column.
6. Repeat step 4&5 for X number of times.
7. Increment DX i.e.row
8. Repeat step 4&7 Y number of lines.
9. Decrement CX i.e.Column.
10. Decrement DX i.e. Row.
12. Repeat step 4 & 9 ‘y’ number of lines.

Post lab Assignments:

1.
2.

Laboratory session 6

Title : To study PCI Bios Interrupt 1AH

References:

1.The Pentium Processor – Tom Shanley

Theory:

Category: **expansion bus BIOSes**

1.INT 1A - PCI BIOS v2.0c+ - FIND PCI DEVICE

```
AX = B102h
CX = device ID (see #00735,#00742,#00743,#00873,#00875)
DX = vendor ID (see #00732)
SI = device index (0-n)
Return: CF clear if successful
CF set on error
AH = status (00h,83h,86h) (see #00729)
    00h successful
        BH = bus number
        BL = device/function number (bits 7-3 device, bits 2-0
func)
EAX, EBX, ECX, and EDX may be modified
all other flags (except IF) may be modified
Notes:  this function may require up to 1024 byte of stack; it
will not enable
  interrupts if they were disabled before making the call
device ID FFFFh may be reserved as a wildcard in future
implementations
the meanings of BL and BH on return were exchanged between the
initial drafts of the specification and final implementation
all devices sharing a single vendor ID and device ID may be
enumerated by incrementing SI from 0 until error 86h is
returned
```

Category: **expansion bus BIOSes**

2.INT 1A - PCI BIOS v2.1+ - SET PCI IRQ

```
AX = B10Fh
BH = bus number
BL = device/function number (bits 7-3 device, bits 2-0
function)
CH = number of IRQ to connect
CL = number of interrupt pin (0Ah=INTA# to 0Dh=INTD#) to
reprogram
DS = segment/selector for PCI BIOS data
real mode: F000h; 16-bit PM: physical 000F0000h; 32-bit PM: as
specified by BIOS32 services directory)
Return: CF clear if successful
    AH = 00h
CF set on error
    AH = error code (59h) (see #01243)
Note:   assumes that the calling application has determined
the IRQ routing
  topology (see AX=B10Eh), has ensured that the selected IRQ
will not
```

```
  cause a conflict, and will update the interrupt line
configuration
  register on all devices which currently use the IRQ line
```

Category: **expansion bus BIOSes**

3.INT 1A - PCI BIOS v2.0c+ - READ CONFIGURATION BYTE

```
AX = B108h
BH = bus number
BL = device/function number (bits 7-3 device, bits 2-0
function)
DI = register number (0000h-00FFh) (see #00878)
Return: CF clear if successful
    CL = byte read
CF set on error
AH = status (00h,87h) (see #00729)
EAX, EBX, ECX, and EDX may be modified
all other flags (except IF) may be modified
Notes:  this function may require up to 1024 byte of stack; it
will not enable
  interrupts if they were disabled before making the call
 the meanings of BL and BH on entry were exchanged between the
initial
  drafts of the specification and final implementation
BUG:    the Award BIOS 4.51PG (dated 05/24/96) incorrectly
returns FFh for
  register 00h if the PCI function number is nonzero
```

Category: **expansion bus BIOSes**

4.INT 1A - PCI BIOS v2.0c+ - READ CONFIGURATION WORD

```
AX = B109h
BH = bus number
BL = device/function number (bits 7-3 device, bits 2-0
function)
DI = register number (0000h-00FFh, must be multiple of 2) (see
#00878)
Return: CF clear if successful
    CX = word read
CF set on error
AH = status (00h,87h) (see #00729)
EAX, EBX, ECX, and EDX may be modified
all other flags (except IF) may be modified
Notes:  this function may require up to 1024 byte of stack; it
will not enable
  interrupts if they were disabled before making the call
 the meanings of BL and BH on entry were exchanged between the
initial
  drafts of the specification and final implementation
BUG:    the Award BIOS 4.51PG (dated 05/24/96) incorrectly
returns FFFFh for
  register 00h if the PCI function number is nonzero
```

5.INT 1A - PCI BIOS v2.0c+ - FIND PCI CLASS CODE

```
AX = B103h
ECX = class code (see also #F0085,#00878)
    bits 31-24 unused
    bits 23-16 class
    bits 15-8  subclass
    bits 7-0   programming interface
SI = device index (0-n)
Return: CF clear if successful
CF set on error
AH = status (00h,86h) (see #00729)
    00h successful
        BH = bus number
        BL = device/function number (bits 7-3 device, bits 2-0
func)
    86h device not found
EAX, EBX, ECX, and EDX may be modified
all other flags (except IF) may be modified
Notes:  this function may require up to 1024 byte of stack; it
will not enable
  interrupts if they were disabled before making the call
 the meanings of BL and BH on return were exchanged between the
initial
  drafts of the specification and final implementation
 all devices sharing the same Class Code may be enumerated by
  incrementing SI from 0 until error 86h is returned
```

Category: **expansion bus BIOSes**

6. INT 1A - PCI BIOS v2.1+ - GET IRQ ROUTING INFORMATION

```
AX = B10Eh
BX = 0000h
DS = segment/selector for PCI BIOS data
    (real mode: F000h; 16-bit PM: physical 000F0000h; 32-bit
PM: as
    specified by BIOS32 services directory)
ES:(E)DI -> IRQ routing table header (see #01259 at AX=B406h)
Return: CF clear if successful
    AH = 00h
    BX = bit map of IRQ channels permanently dedicated to PCI
    WORD ES:[DI] = size of returned data
CF set on error
    AH = error code (59h) (see #01243)
 WORD ES:[DI] = required size of buffer
```

7. INT 1A - PCI BIOS v2.0c+ - WRITE CONFIGURATION BYTE

```
AX = B10Bh
BH = bus number
BL = device/function number (bits 7-3 device, bits 2-0
function)
DI = register number (0000h-00FFh)
CL = byte to write
Return: CF clear if successful
CF set on error
AH = status (00h,87h) (see #00729)
EAX, EBX, ECX, and EDX may be modified
```

```
   all other flags (except IF) may be modified
  Notes:  this function may require up to 1024 byte of stack; it
  will not enable
     interrupts if they were disabled before making the call
   the meanings of BL and BH on entry were exchanged between the
  initial
     drafts of the specification and final implementation
```

Laboratory Session 7

Title: To write a program to demonstrate

1. Task scheduling in Embedded System using MicroC-OS II Simulator
2. Usage of IPC in system design

References: 1. RajKamal , Embedded Systems –Architecture, Programming and Design
2. Sriram Iyer and Pankaj Gupta : Embedded Real Time Systems

Pre-Requisites: 1. Knowledge of C and Assemble language
2. Concepts of different types of task Scheduling
3. Micro-OS II functions

Theory: Medium and Sophisticated Embedded systems are generally real time constrained and hence need to perform multiple tasks. Multi-Tasking enables the system to perform tasks by switching between them efficiently. These systems generally consist of an RTOS which helps in managing the various application tasks. There are generally two ways to schedule tasks in any system

1. Pre-Emptive
2. Non Pre-Emptive.

Depending on the nature of the system and the tasks , either method can be used. In a Pre-Emptive system, all the tasks are assigned a priority, generally at design time. At any instant, if a low priority task is running and a high priority task need to run, the RTOS pre-empts the low priority task and hands over the CPU to the higher one. However, in Non Pre- emptive systems, priorities are not assigned and methods like time-slicing, Round Robin Scheduling, etc are used to service the tasks.

Task Synchronization may be required in systems where two independent tasks need to be synchronized. Most RTOSes offer methods for Inter-Process Communication (IPC) like Semaphores, Message Queues, Mailboxes, etc.

MicroC/OS-II

MicroC/OS-II(commonly termed **µC/OS-II** or **uC/OS-II**), is the acronym for Micro-Controller Operating Systems Version 2. It is a priority-based pre-emptive real-time multitasking operating system kernel for microprocessors, written mainly in the C programming language. It is intended for use in embedded systems. Its features are:

- It is a very small real-time kernel.
- Memory footprint is about 20KB for a fully functional kernel.
- Source code is written mostly in ANSI C.
- Highly portable, ROMable, very scalable, preemptive real-time, deterministic, multitasking kernel.

- It can manage up to 64 tasks (56 user tasks available).
- It has connectivity with μC/GUI and μC/FS (GUI and File Systems for μC/OS II).
- It is ported to more than 100 microprocessors and microcontrollers.
- It is simple to use and simple to implement but very effective compared to the price/performance ratio.
- It supports all type of processors from 8-bit to 64-bit.

Algorithm:

1. Pre-Emptive task scheduling

1. Start
2. Initialise OS
3. Create Task 1 and Task 2 (Assign priorities){ Task 1 and Task 2 display different strings}
4. Start OS
5. End

2. IPC (Usage of semaphore)

1. Start
2. Initialize OS
3. Create Task1 and Task 2 { Task 1 posts semaphore and Task 2 pends for the semaphore}
4. Create Semaphore
5. Start OS
6. End
7.

Post Laboratory Questions:

1. Explain different Non Pre-Emptive scheduling methods
2. Explain the different forms of semaphore related functions supported in MicroC-OSII.